
UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería del Software

# Intérprete del lenguaje de signos en tiempo real mediante Machine Learning

# Sign language recognition in real time with Machine Learning

Realizado por
Daniel Gallego Peralta

Tutorizado por
David Santo Orcero

Departamento
Lenguajes y ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, junio 2021

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Graduado en Ingeniería del Software

# Intérprete del lenguaje de signos en tiempo real mediante Machine Learning

# Sign language recognition in real time with Machine Learning


Realizado por
**Daniel Gallego Peralta**

Tutorizado por
**David Santo Orcero**

Departamento
**Lenguajes y ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2021


Fecha defensa: 7 de julio de 2021

# Abstract

Development of a mobile application for iOS which will be able to recognize in real time letters from American sign Language. The user will select the camera of the device or a video as an input and the application will try to detect the hand of a person, in case of success, the application will perform a prediction about which letter the hand is representing. The mobile application is developed in Swift and it is using a Deep Learning model for the prediction. The development of a model is using Keras as a framework for the generation and training of the model. The model consists of a neural network that will be able to perform predictions about 24 letters of the American Sign Language(ASL), excluding letters whose representation is not a static gesture. The model is developed for the use in mobile devices, especially iOS. The application uses ASL as a language to detect because there are multiple public datasets of images available, and these datasets are used for the training of the model. But the adaptation to the LSE(Spanish Sign Language) is simple, with a good dataset of images about LSE the project can generate a new model for the application to use it.

**Keywords:** **iOS, Machine Learning, Sign Language, Swift, Keras**

# Resumen

Desarrollo de una aplicación móvil para iOS, que es capaz de identificar en tiempo real letras de la lengua de signos americana(ASL). El usuario podrá seleccionar la cámara del dispositivo o un video y la aplicación intentará detectar las manos de una persona, en caso de éxito, realizará una predicción para identificar qué letra está realizando con la mano.

La aplicación será realizada en Swift y utiliza un modelo de Deep Learning para las predicciones. Para el desarrollo del modelo, se utiliza Keras como una librería para la generación y el entrenamiento. Consistirá en una red neuronal que será capaz de realizar predicciones sobre 24 letras, excluyendo letras cuya representación no será un gesto estático, y que está desarrollada para su uso en dispositivos móviles, especialmente iOS.

Se utiliza como lenguaje ASL especialmente por la multitud de conjuntos de imágenes que son públicos y que se utilizan para el entrenamiento del modelo. Si bien su adaptación a LSE (Lengua de Signos Española) consistirá simplemente en su adaptación a un conjunto de imágenes del lenguaje de signos en español, puesto que el proyecto contiene todas las herramientas para su adaptación a otro lenguaje.

**Palabras clave:** **iOS, Machine Learning, Sign Language, Swift, Keras**

# Índice

# 1 Introducción

## 1.1. Motivación

Este proyecto surge para investigar el uso de algoritmos de Deep Learning en dispositivos móviles. Por este motivo, surge como una investigación de cara al futuro, cuyo objetivo es investigar la capacidad de trabajar con este tipo de algoritmos en los dispositivos móviles iOS de última generación. El proyecto se enfoca en la interpretación del lenguaje de signos ASL. Si bien es una primera toma de contacto para su interpretación, se trata de ver si es posible su interpretación bajo las capacidades de un dispositivo móvil y si puede realizarse en un tiempo de respuesta suficientemente rápido como para actuar como un intérprete.

ASL es un lenguaje dinámico que es expresado mediante el movimiento de las manos y la cara. Se plantea el proyecto como un primer paso para lograr este objetivo final. El proyecto identificará 24 letras del alfabeto realizado con ASL. Se excluyen las letras cuya representación requiere de un gesto dinámico. Si bien la motivación es poder ser capaz de interpretar gestos que simbolizan palabras o frases, el proyecto se plantea solamente como un primer paso para ello.

La principal motivación del proyecto surge para resolver los dos siguientes casos de uso:

1. Interpretación del lenguaje de signos: una persona sin conocimientos sobre ASL, necesita entender a otra que se está comunicando mediante este lenguaje, la aplicación se encargará de interpretar y comunicar el lenguaje de signos.

2. Investigación sobre las capacidades de los dispositivos móviles en la ejecución de algoritmos de Machine Learning: Demostrar que los dispositivos iOS de última generación pueden resolver este tipo de problemas en tiempo real y que lo aprendido en este proyecto puede ser igualmente aplicado a cualquier tipo de aplicación que se enfoque en detectar que está haciendo una persona.

## 1.2. Objetivos

El objetivo del proyecto es tener un sistema completo que va desde el desarrollo de un modelo de Deep Learning y su entrenamiento, exportación del modelo para su uso en dispositivos iOS, y tener una aplicación móvil que use este modelo como base para la interpretación del lenguaje de signos. Se utilizarán Datasets públicos del ASL para el entrenamiento del modelo, pero el objetivo del proyecto es que el sistema sea fácilmente adaptable a otros lenguajes. Para su consecución, se consideran los siguientes objetivos:

- Proporcionar todas las componentes del sistema para que, con otro Dataset de un lenguaje diferente, se genere todo el proceso para tener una aplicación lista para interpretar ese lenguaje.
- La aplicación admitirá diferentes tipos de entrada: la cámara del dispositivo, selección de videos desde la librería del teléfono o iCloud y mostrará por pantalla la identificación de letras.
- El proyecto se plantea como una investigación sobre la capacidad de los dispositivos iOS para interpretar este tipo de algoritmos en tiempo real e intentar exprimir al máximo sus capacidades, por lo que sólo se considerará su buen funcionamiento únicamente en dispositivos de última generación como el iPhone 12 Pro.
- El modelo de Deep Learning está desarrollado específicamente para su uso en dispositivos móviles iOS y en tiempo real. Por lo tanto, aspectos como su peso y rapidez son fundamentales. Se utilizará Keras para la generación y entrenamiento del modelo.
- Privacidad y consumo de datos: Todo el procesamiento de datos se llevará a cabo en el dispositivo sin la ayuda de un servidor externo, de forma que se asegura la total privacidad del usuario y el uso mínimo de la red.

## 1.3. Tecnologías usadas

Para el desarrollo del proyecto se utilizan los siguientes lenguaje de programación y tecnologías:

- Swift: Lenguaje de programación para el desarrollo de la aplicación móvil.
- Vision y CoreML: Frameworks iOS utilizados para el reconocimiento de imágenes y Machine Learning.
- Python, Keras: Python es el lenguaje de programación utilizado junto con Keras como framework para la generación y entrenamiento de un modelo de Machine Learning
- Django: Se utilizará Django como API REST en una parte del proyecto.

## 1.4. Estructura del documento

La memoria se compone de los siguientes capítulos:

1. Introducción: Desarrollo de los objetivos, motivación, tecnologías usadas y explicación de la estructura del proyecto.
2. Bases del proyecto: Explicación de las bases del proyecto y los diferentes componentes necesarios para el funcionamiento del sistema completo.
3. API REST: Proporcionará un api para obtener las imágenes del dataset.
4. Extracción de características: Desarrollo del componente encargado de la extracción de características que van a ser usadas para el entrenamiento de un modelo de Deep Learning.
5. Modelo: Desarrollo del modelo de Deep Learning, así como su entrenamiento y su exportación a un formato compatible con iOS.
6. Intérprete: Desarrollo de la aplicación móvil para iOS encargada de la interpretación del lenguaje de signos.
7. Resultados y pruebas de las diferentes configuraciones utilizadas.
8. Conclusiones y líneas futuras.

# 2

# Bases del proyecto

## 2.1. Lenguaje de signos

La aplicación será capaz de interpretar la lengua de signos americana (ASL). Dentro de esta lengua, se encargará de poder interpretar el alfabeto de letras.

Dentro del alfabeto, se excluyen dos letras: la 'Z' y la 'J'. Debido a que su representación se realiza con un gesto, y no puede realizarse de forma estática. Por lo tanto, como se puede ver en la Figura 1, estas son las diferentes letras que podrá interpretar la aplicación.

## 2.2. Extracción de características

Para la generación de un modelo que sea capaz de interpretar signos, la parte más importante es la extracción de características. Normalmente la extracción de características se realiza como paso previo al entrenamiento del modelo, y es realizado por el mismo componente que se encarga de compilar y entrenar el modelo. Pero en este proyecto se va a realizar en otro componente del sistema.

Las características que se van a utilizar para entrenar el modelo están basadas en las posiciones relativas de los dedos. Para ello, tendremos que detectar en una imágen donde está cada uno de los dedos, de forma que podamos obtener coordenadas para diferentes posiciones de los dedos y en base a estas coordenadas entrenar un modelo.

Este proceso de reconocimiento de imágenes con tanta precisión es un problema bastante complejo de resolver, pero dado que este proyecto es una aplicación para dispositivos iOS, se va a utilizar Apple Vision Framework para resolver este problema.

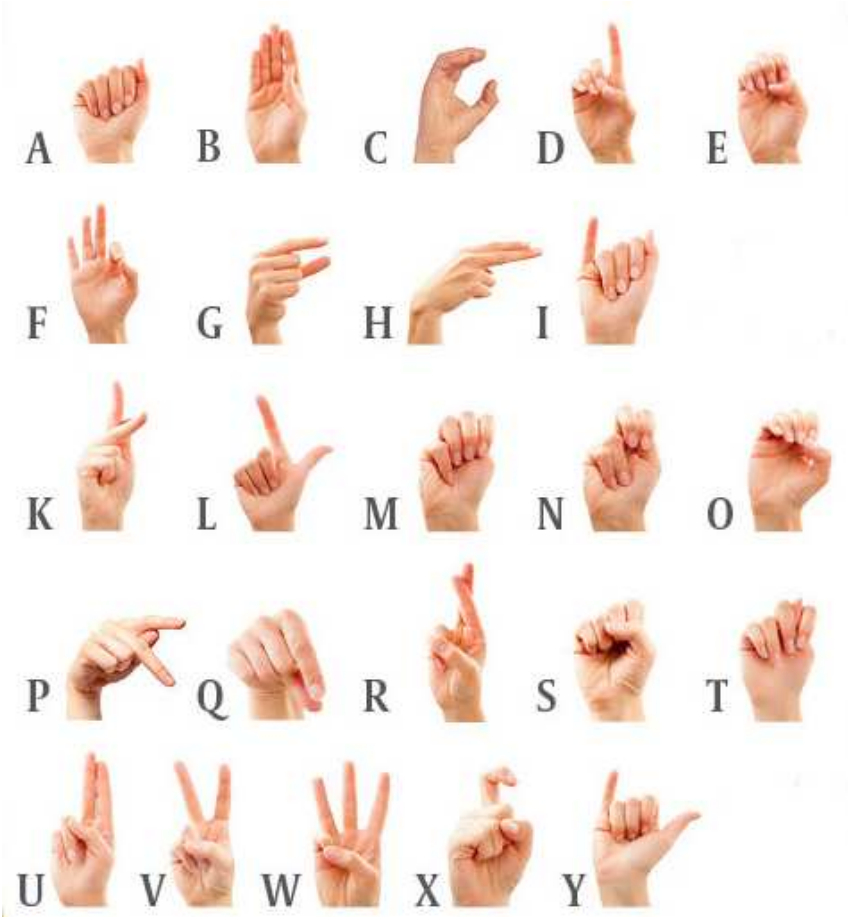



Figura 1: Imágenes de las 24 letras con su gesto en ASL Recuperado de: https://deafchildren.org/2019/06/free-asl-alphabet-chart/

Apple Vision Framework: The Vision framework performs face and face landmark detection, text detection, barcode recognition, image registration, and general feature tracking. Vision also allows the use of custom Core ML models for tasks like classification or object detection [4]

Con el uso de Vision Framework, podremos detectar si existe una mano en una imagen, y a su vez detectar las posiciones relativas de cada uno de los dedos. Utilizamos estos datos como entrada para el entrenamiento de un modelo de Deep Learning realizado en Keras.

## 2.3. Modelo Deep Learning

Se construye un modelo de Deep Learning que utiliza como entrada los datos generados por Vision Framework para entrenar el modelo. Se utiliza Python y Keras como herramien-

tas. El modelo es una red neuronal con diferentes capas que tiene como entrada 42 valores, representando diferentes coordenadas de los dedos de una mano, y tendrá como salida una distribución de probabilidad que indica la probabilidad de que la entrada pertenezca a cada una de las 24 letras que estamos intentando identificar.

Se realiza un proceso de entrenamiento y mejora del modelo, mediante Fine-Tuning de hiper-parámetros, y como último paso tendremos que convertir el modelo generado a un formato compatible con dispositivos iOS: Core ML, para ello se utiliza la herramienta Core ML Tools [10].

Core ML: Core ML is an Apple framework to integrate machine learning models into your app. Core ML provides a unified representation for all models. Your app uses Core ML APIs and user data to make predictions, and to fine-tune models, all on the user's device. Core ML optimizes on-device performance by leveraging the CPU, GPU, and Neural Engine while minimizing its memory footprint and power consumption. Running a model strictly on the user's device removes any need for a network connection, which helps keep the user's data private and your app responsive [9]

## 2.4. Dataset

Se utilizan diferentes datasets para la extracción de características. A su vez, conforme avanza el entrenamiento y se obtienen los resultados en las diferentes predicciones de las letras, se revisan aquellas imágenes en las que el modelo está teniendo problemas para acertar en sus predicciones. Por lo tanto, el dataset final será una mezcla de los siguientes datasets tras un proceso de refinado para obtener los mejores resultados.

Se han utilizado los siguientes dataset:

1. Image data set for alphabets in the American Sign Language [5].
2. ASL Alphabet Images with a variety of backgrounds for validating a model [6].
3. Significant (ASL) Sign Language Alphabet Dataset [22].
4. ASL Sign Language Alphabet Pictures [Minus J, Z] [7].
5. American Sign Language Alphabet (Static)] [2].

6. Dataset propio con algunas imágenes

## 2.5. API REST

Se utiliza un proyecto realizado con Python y Django que servirá para almacenar los diferentes datasets, así como proporcionar acceso vía web a la imágenes de cada uno. También proporcionará un API REST que permite obtener todos los enlaces a la imágenes organizados por letras.

## 2.6. Training App

Para la extracción de características se utiliza Vision Framework, este framework está solamente disponible en dispositivos iOS, por lo que se realiza una aplicación iOS cuya utilidad es exclusivamente la extracción de características.

Esta aplicación conectará con el API REST para obtener las imágenes a analizar, realizará una extracción de características de las imágenes, y proporcionará los resultados en formato .csv. Este producto se realizará en una app diferente porque no debe formar parte del producto principal, que debe centrarse únicamente en interpretar ASL.

## 2.7. ASL Interpreter App

Será el producto final y la aplicación que utilizará el usuario final. Utiliza el Modelo de Deep Learning generado por otro componente y su función es permitir diferentes entradas como puede ser usar de la cámara del dispositivo o seleccionar vídeos, y realizar predicciones. Las predicciones obtenidas por el modelo, pasarán por un proceso de comparar predicciones consecutivas para obtener resultados más fiables, y mostrará por pantalla los resultados obtenidos.

# 3
# API REST

## 3.1. Django

Se utiliza Django para proporcionar un API REST que da acceso a las imágenes del dataset. Así como proporcionar un listado de las url de las imágenes organizadas por letras. Este listado será usado por la aplicación de entrenamiento para extraer las características de las imágenes.

Para la ejecución del proyecto se utiliza Pycharm [20], que se encarga de la compilación y ejecución en un servidor web local. También se utiliza Django REST Framework [19] para facilitar la creación de un endpoint. Se proporciona el código completo del proyecto, así que se detallan solamente los aspectos más importantes:

- Ejecución en un entorno local: Se utiliza el IDE Pycharm para ejecutar el proyecto, para ello se establece la siguiente configuración 2

- Settings.py: Configuración de la ruta donde se encuentran las imágenes del dataset.

```
STATIC_URL = '/static/'
DEFAULT_SCHEME = "http://192.168.1.5:8000"

MEDIA = 'media'
DATASET = 'dataset'

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, "media")

MEDIA_DEFAULT = DEFAULT_SCHEME + MEDIA_URL
DATASET_ROOT_DIR = os.path.join(MEDIA_ROOT, 'dataset')
```

```
DATASET_URL = DEFAULT_SCHEME + MEDIA_URL + DATASET + "/"
```

- Estructura de la carpeta media: Contiene el conjunto de datasets utilizados en el proyecto, están numerados del 1 al 6 y dentro de cada uno se organizan por letras 3.

- Obtener un listado de las imágenes: Proporcionar un endpoint para obtener en formato JSON un listado de todas las imágenes organizadas por letras.

```
 class ImagesListAPI(APIView):
""" List of urls images by letters"""
def get(self, request):
    labelsClass = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K",
                   "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U",
                   "V", "W", "X", "Y"]
    json_data = {}
    train_folder = os.path.join(settings.DATASET_ROOT_DIR, 'train')
    for letter in labelsClass:
        list_images = []
        for i in range(1, 7):
            folder_data = os.path.join(train_folder, str(i), letter)
            images_list = os.listdir(folder_data)
            for imageName in images_list:
                url = settings.DATASET_URL + "train" + "/" + str(i) +
"/" + letter + "/" + imageName
                list_images.append(url)
        json_data[letter] = list_images
    return JsonResponse(json_data)
```

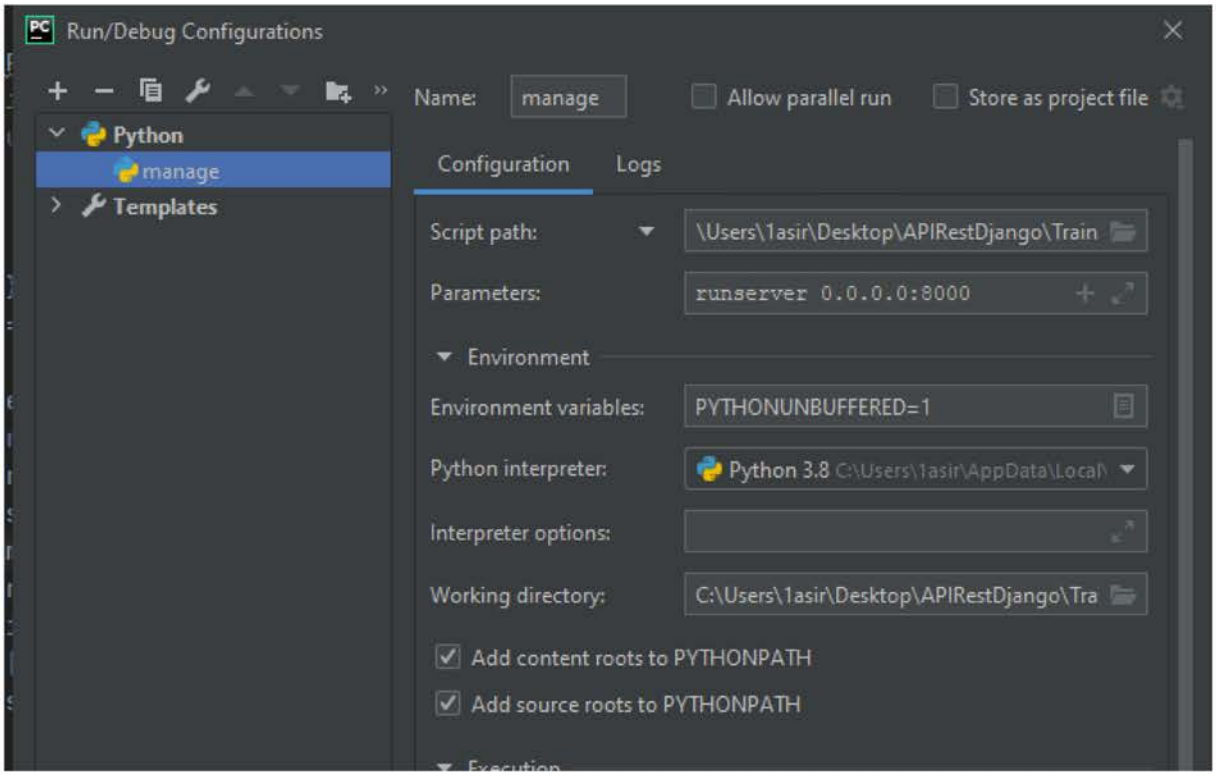



Figura 2: Configuración de ejecución en Pycharm

El resultado de este componente del sistema se muestra en la figura 4 y será usado por la app de entrenamiento para obtener las imágenes a analizar. Tener este componente organizado por diferentes dataset numerados nos permite realizar pruebas realizando entrenamientos sobre sólo una selección de ellos y realizar modificaciones sobre cuáles de los dataset se van a incluir en el entrenamiento del sistema.


Figura 3: Organización de la carpeta media

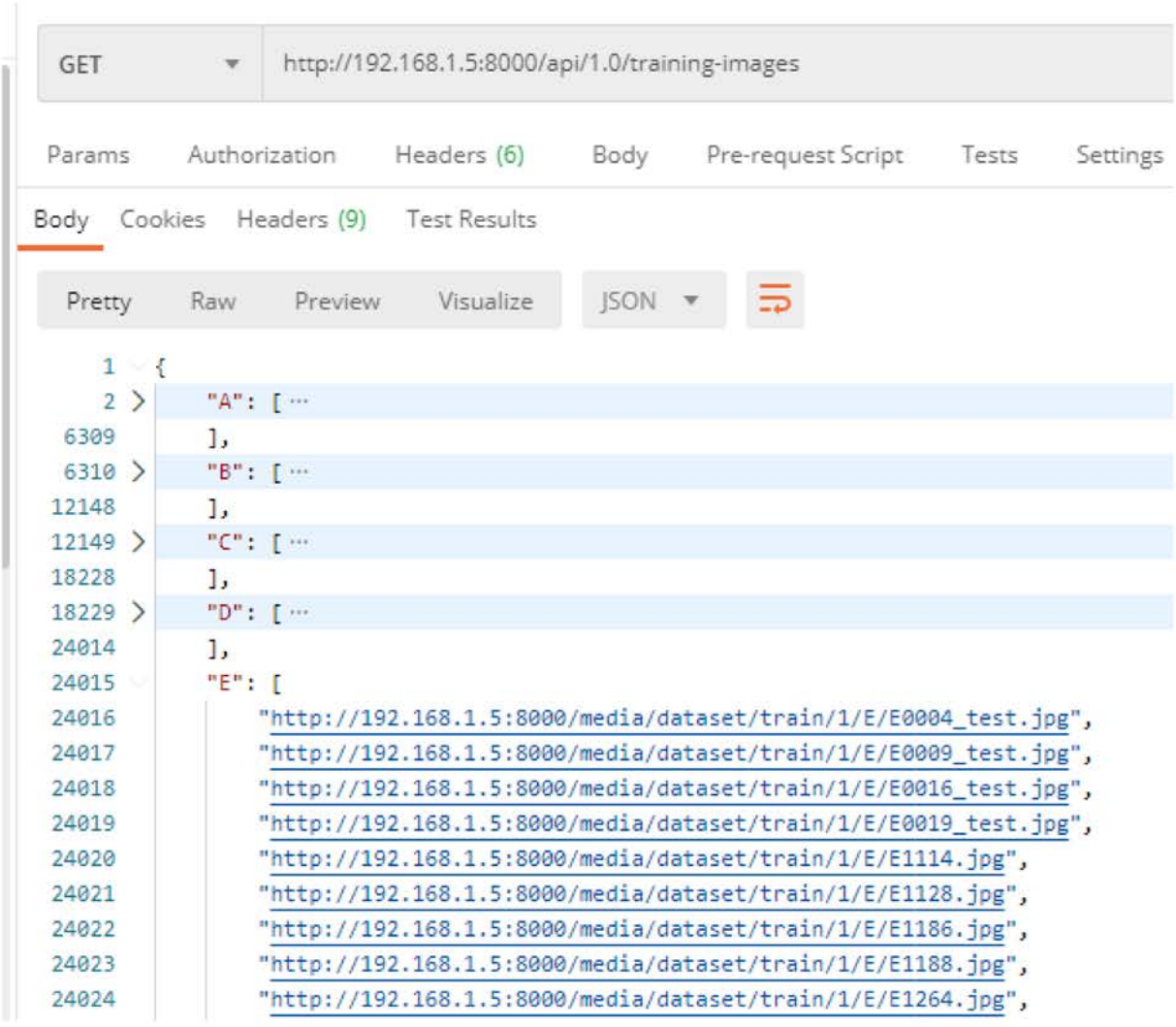



Figura 4: Listado en formato JSON de las imágenes a analizar organizadas por letras.

# 4

# Extracción de características

## 4.1. Overview

Se desarrolla una aplicación móvil para iOS para la extracción de características que serán usadas en el entrenamiento de un modelo de Deep Learning.

Está desarrollada en Swift 5 y será compatible con iPhone. El caso de uso de la aplicación será el siguiente:

La aplicación realizará una llamada a un endpoint del API REST en un entorno local y obtendrá como respuesta un listado en formato JSON de las imágenes del dataset. El listado consiste en un diccionario en el cual por cada letra que se va a interpretar tendrá un listado de imágenes que representan esa letra.

Por cada url obtenida, se descarga la imagen y se utilizará Vision Framework para obtener una predicción que consiste en detectar si existe una mano en la imagen, en caso de que exista, obtendremos como resultado las coordenadas de varias posiciones de la mano. Estas coordenadas incluyen la posición de la muñeca, y 4 posiciones de cada uno de los dedos, desde la punta del dedo hasta la parte inferior.

Para cada conjunto de coordenadas que representan las posiciones de los dedos, se realiza un proceso de normalización. Este proceso persigue dos objetivos:

1. Lo importante de la posición de la mano no es en qué punto de la imagen está localizada, no importa si está en la parte superior de la imagen o en la parte inferior, sino lo importante es la posición relativa entre los dedos. Por tanto, el objetivo es que la misma posición de los dedos, sin importar en qué posición de la imagen esté localizada, tenga las mismas coordenadas.

2. Evitar que las coordenadas sean números demasiado pequeños porque dificultará el entrenamiento de un modelo de Deep Learning. Vamos a normalizar las coordenadas de la mano entre los valores 0.0 y 1.0, Siempre tendremos un punto de la mano que representará el valor 0.0 y otro punto de la mano que representará el valor 1.0, para ambos ejes de coordenadas.

Una vez procesadas todas las imágenes y obtenidas sus características, se procede a almacenarlas en disco en ficheros con formato csv. La organización de los ficheros se muestra en la figura 5:

Por último, se muestra la opción de compartir los resultados obtenidos, se utiliza esta función para comprimir y descargar los resultados obtenidos. Se podrán descargar por email, Airdrop o cualquier otra aplicación compatible en iOS.

## 4.2. Arquitectura

La aplicación seguirá la arquitectura MVVM: Model-View-ViewModel. Constará de una única pantalla y la aplicación cuenta con los siguientes componentes:

1. ExtractFeaturesPicturesVC, ExtractFeaturesPicturesView: Se encargará de la interfaz, se comunica únicamente con el ViewModel.

2. ExtractFeaturesPicturesViewModel: Será el ViewModel de la aplicación. Se encarga de comunicarse con diferentes workers que realizarán el trabajo de procesamiento de las imágenes y ficheros, y comunicará los resultados a la vista.

3. Hand, Finger: Son los modelos de la aplicación. Ambos realizarán el proceso de convertir los resultados obtenidos a través de Vision framework en las coordenadas normalizadas que van a ser usadas para el entrenamiento del modelo.

4. VisionProcessImage, ApiManager, FileWorker: se encargan de realizar el procesamiento y coordinación de las operaciones a realizar, desde comunicarse con el API REST hasta almacenar los resultados obtenidos.

## 4.3. Interfaz

La interfaz se puede encontrar en los siguientes estados:

1. Primer lanzamiento de la aplicación. La única opción posible es empezar con la extracción de las características 6.

2. Extracción de las características en proceso 7.

3. Tras finalizar la extracción de características podemos descargar los resultados por diferentes medios 8.

4. En un lanzamiento posterior, podemos empezar otro entrenamiento o descargar los resultados de una ejecución anterior 9.

## 4.4. VNDetectHumanHandPoseRequest

El código completo del proyecto es bastante amplio así que este documento se centra en las dos partes más importante que es la detección y normalización de coordenadas de los dedos de una mano. Para la detección de los dedos, se utiliza Vision Framework, y más específicamente, se utiliza VNDetectHumanHandPoseRequest [1].

VNDetectHumanHandPoseRequest detectará si existe una mano en una imagen, y en caso de que exista, se obtienen los siguientes resultados.

Como se puede ver en la figura 10, podemos obtener un total de 21 posiciones de la mano, 4 por cada uno de los dedos y una por la muñeca. Para cada posición, tendremos dos coordenadas representando cada eje, este nos deja con 42 coordenadas, que serán las características que usaremos para entrenar el modelo.

En las figuras 11, 12, 13, se muestra con más detalle las diferentes posiciones que se obtienen como resultado, las imágenes y más información relativa a VNDetectHumanHandPoseRequest puede encontrarse en la siguiente documentación [18].

Tras aplicar VNDetectHumanHandPoseRequest a una imagen, obtendremos una objeto de tipo VNHumanHandPoseObservation, del que extraemos las coordenadas para cada una

de 21 posiciones de los dedos y muñeca. La siguiente función se encarga de convertir VNHumanHandPoseObservation en un objeto HandFingerPoints que se encarga de la normalización de coordenadas.

```
func processObservations(_ observation: VNHumanHandPoseObservation) -> Hand? {
    guard observation.confidence ≥ minConfidenceObservation else {
        print("observation low confidence")
        return nil
    }

    do {
        let wrist = try observation.recognizedPoint(.wrist)
        let indexFingerPoints = try observation.recognizedPoints(.indexFinger)
        let ringFingerPoints = try observation.recognizedPoints(.ringFinger)
        let thumbFingerPoints = try observation.recognizedPoints(.thumb)
        let littleFingerPoints = try observation.recognizedPoints(.littleFinger)
        let middleFingerPoints = try observation.recognizedPoints(.middleFinger)

        let hand = HandFingerPoints(indexFingerPoints: indexFingerPoints,
                                    ringFingerPoints: ringFingerPoints,
                                    littleFingerPoints: littleFingerPoints,
                                    middleFingerPoints: middleFingerPoints,
                                    thumbFingerPoints: thumbFingerPoints,
                                    wristPoint: wrist)
        hand.normalizeHand()
        numberOfProcess += 1
        return hand
    } catch {
        print(error)
        return nil
    }
}
```

## 4.5. HandFingerPoints

Es el modelo que representa una mano. Se inicializa con las coordenadas de 21 posiciones de la mano, y se encarga de normalizar estas coordenadas y devolver los datos que serán almacenados como una interpretación de una letra formada por 42 números entre 0.0 y 1.0.

Cada posición está representada por una coordenada X y una coordenada Y. El proceso de normalización de coordenadas cuenta con dos fases para cada una de las coordenadas:

- Transformar a coordenadas relativas: En el caso de la coordenada X, detectamos cual es el mínimo de todas las coordenadas X de la mano, este punto será el punto 0.0, y al resto de las coordenadas le restamos la diferencia entre el punto 0.0 original de la imagen y el que se ha establecido. De esta forma, no importa donde esté situada la mano en la imagen, lo que importa es cómo se relacionan las posiciones de los dedos entre sí. Se aplica para las coordenadas X e Y detectando cada una su mínimo en su eje. Esta será la función encargada de ello para cada dedo (4 puntos por cada dedo):

```
func translatesToRelative(minX: Double, minY: Double) {
    if let point = tip {
        let x = point.x - minX
        let y = point.y - minY
        self.tip = VNPoint(x: x, y: y)
    }

    if let point = dip {
        let x = point.x - minX
        let y = point.y - minY
        self.dip = VNPoint(x: x, y: y)
    }

    if let point = pip {
        let x = point.x - minX
        let y = point.y - minY
        self.pip = VNPoint(x: x, y: y)
    }
```

```
    if let point = mcp {
        let x = point.x - minX
        let y = point.y - minY
        self.mcp = VNPoint(x: x, y: y)
    }
}
```

- Normalizar entre 0.0 y 1.0: En el caso de la coordenada X, la menor posición de todos los puntos siempre tendrá el valor 0.0, y la mayor posición el valor 1.0. Esto incrementa las diferencias entre los dedos de forma que puedan ser más visibles para el modelo de Deep Learning. Para ello, se detectan los mínimos y los máximos de cada eje, se calculan las diferencias entre ellos, y a cada punto se le resta el mínimo de sus eje y se divide por la diferencia entre el máximo y el mínimo. Se puede ver con más detalle en la siguiente función, que se encarga de normalizar las coordenadas de un dedo:

```
func normalizeFinger(minX: Double, minY: Double, maxX: Double, maxY: Double)
    {
    let divideX = maxX - minX
    let divideY = maxY - minY

    if let point = tip {
        let x = (point.x - minX) / divideX
        let y = (point.y - minY) / divideY
        self.tip = VNPoint(x: x, y: y)
    }

    if let point = dip {
        let x = (point.x - minX) / divideX
        let y = (point.y - minY) / divideY
        self.dip = VNPoint(x: x, y: y)
    }
```

```
    if let point = pip {
        let x = (point.x - minX) / divideX
        let y = (point.y - minY) / divideY
        self.pip = VNPoint(x: x, y: y)
    }

    if let point = mcp {
        let x = (point.x - minX) / divideX
        let y = (point.y - minY) / divideY
        self.mcp = VNPoint(x: x, y: y)
    }
}
```

Tras este proceso de normalización de coordenadas, la última función de este modelo es devolver un vector de tipo Double que esté formado por 42 valores entre 0.0 y 1.0 que representan una letra. Este vector será almacenado en un fichero, en el que cada línea del fichero contiene una lista de 42 valores que representan una letra. Por cada letra, tendremos un fichero en formato csv en el que cada línea está formada por 42 valores que representan la misma letra tomada de una imagen diferente. Estos serán los datos que utilizaremos para entrenar un modelo de Deep Learning.

Se muestra una parte del código en la siguiente función, en el que se extraen las coordenadas normalizadas de cada uno de los dedos así como de la muñeca, y en el caso de que todos los valores serán correctos, se forma un vector con todos ellos:

```
/// extract the location of all position in the hand if there is not null data
/// - Returns: 42 values array with the relatives coordinates of each position of
    the hand
func extractIfValidFeatures() -> [Double]? {
    guard let featuresIndex = indexInfo.extractIfValidFeatures(),
          let featuresRing = ringInfo.extractIfValidFeatures(),
          let featuresLittle = littleInfo.extractIfValidFeatures(),
          let middleInfo = middleInfo.extractIfValidFeatures(),
          let thumbInfo = thumbInfo.extractIfValidFeatures(),
```

```
          let wristInfo = wrist.extractIfValidFeatures() else {
        print("No valid hand")
        return nil
    }

    var result = [Double]()
    result.append(contentsOf: featuresIndex)
    result.append(contentsOf: featuresRing)
    result.append(contentsOf: featuresLittle)
    result.append(contentsOf: middleInfo)
    result.append(contentsOf: thumbInfo)
    result.append(contentsOf: wristInfo)
    return result
}
```

Parte de este código será también usado en la aplicación para interpretar ASL, en la aplicación de entrenamiento se utiliza este modelo para obtener las características que serán usadas para entrenar un modelo de Deep Learning. En la aplicación para interpretar ASL, estos datos serán la entrada del modelo de Deep learning, el cual nos devolverá una predicción sobre los datos de entrada.

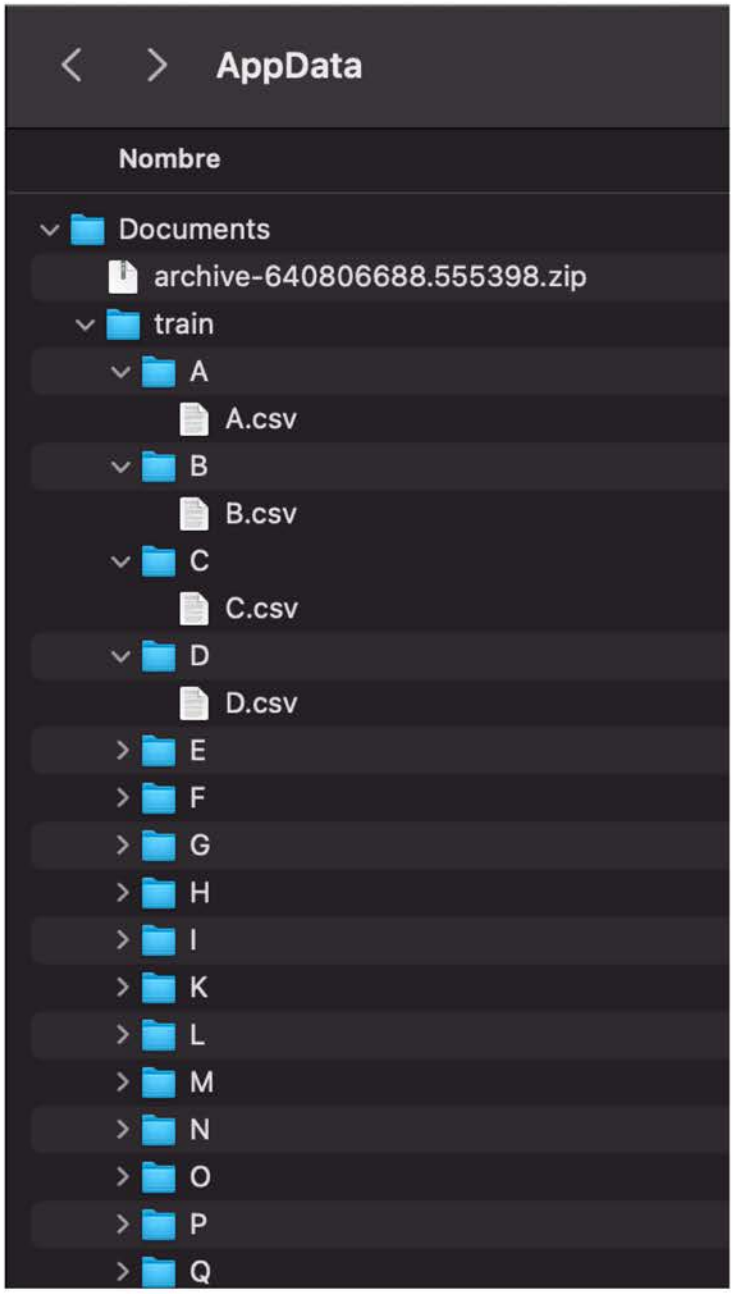



Figura 5: Estructura de los resultados de la extracción de características.

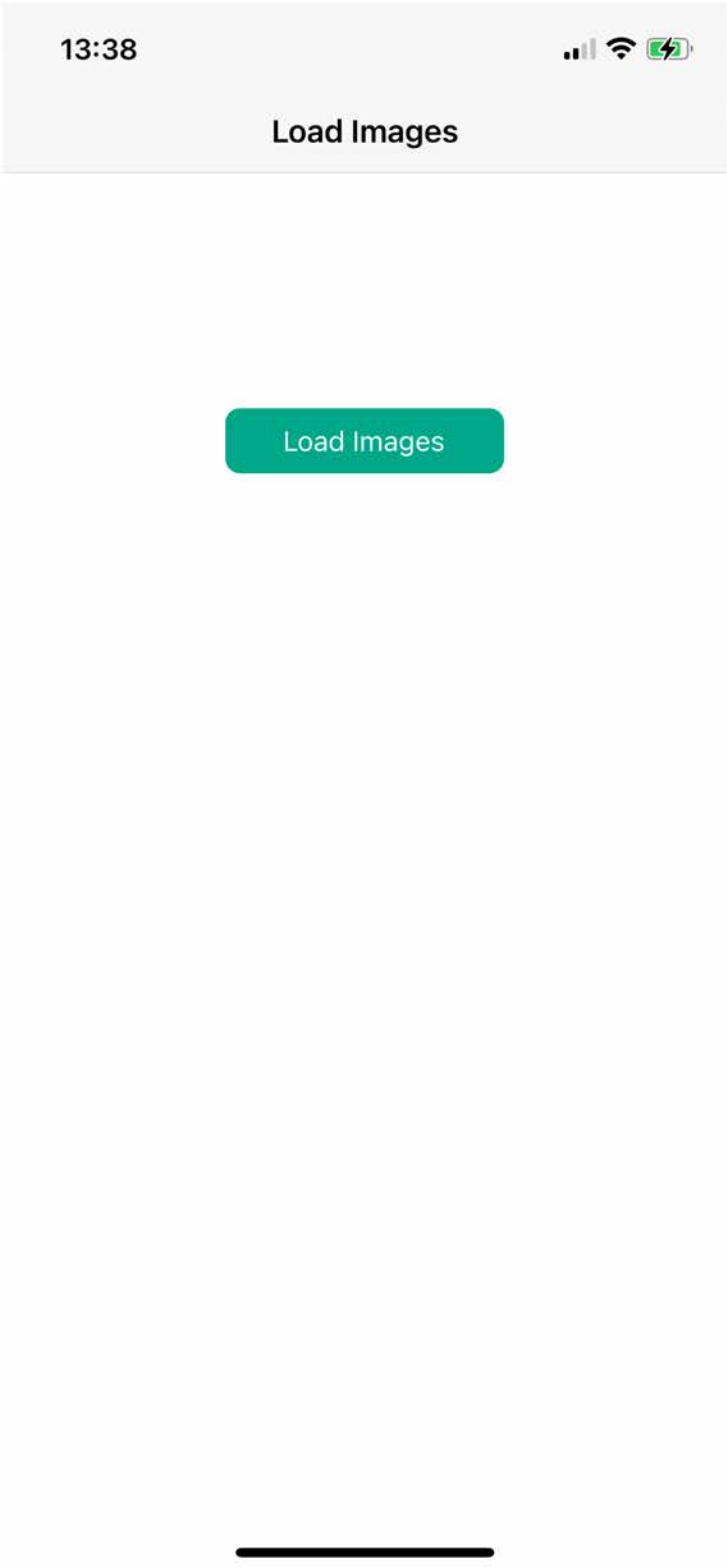



Figura 6: Primer lanzamiento de la aplicación.

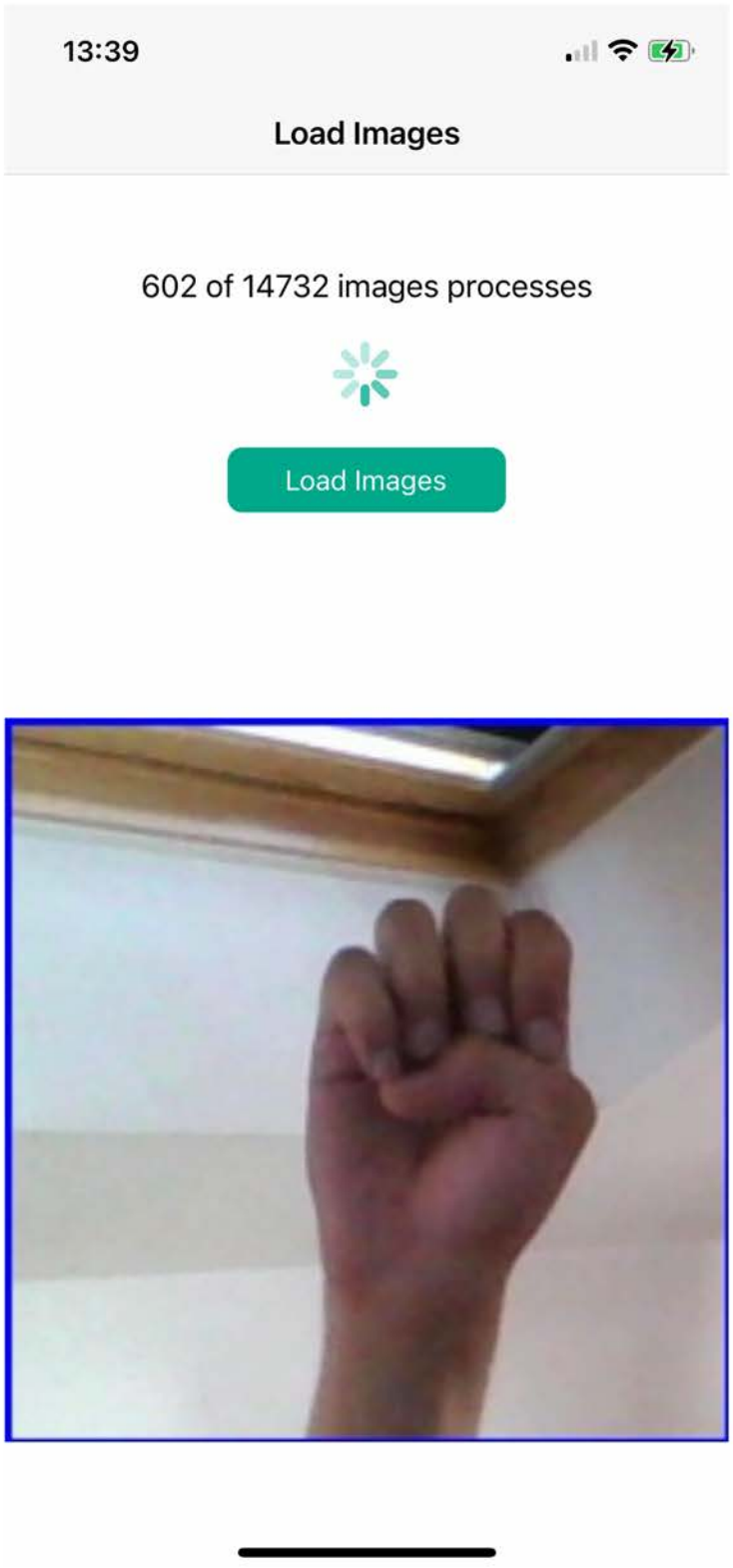



Figura 7: Entrenamiento en proceso.

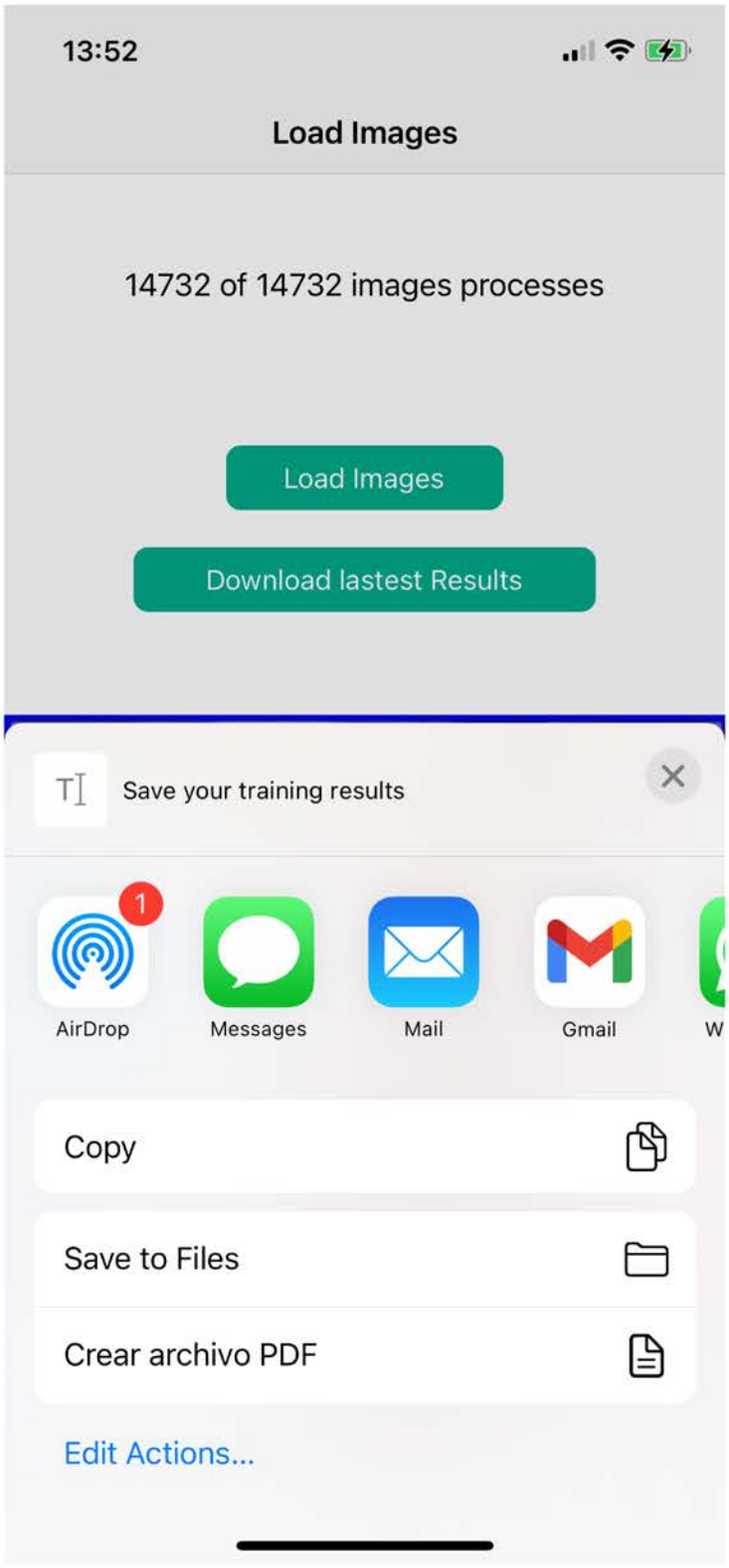



Figura 8: Descargar resultados.

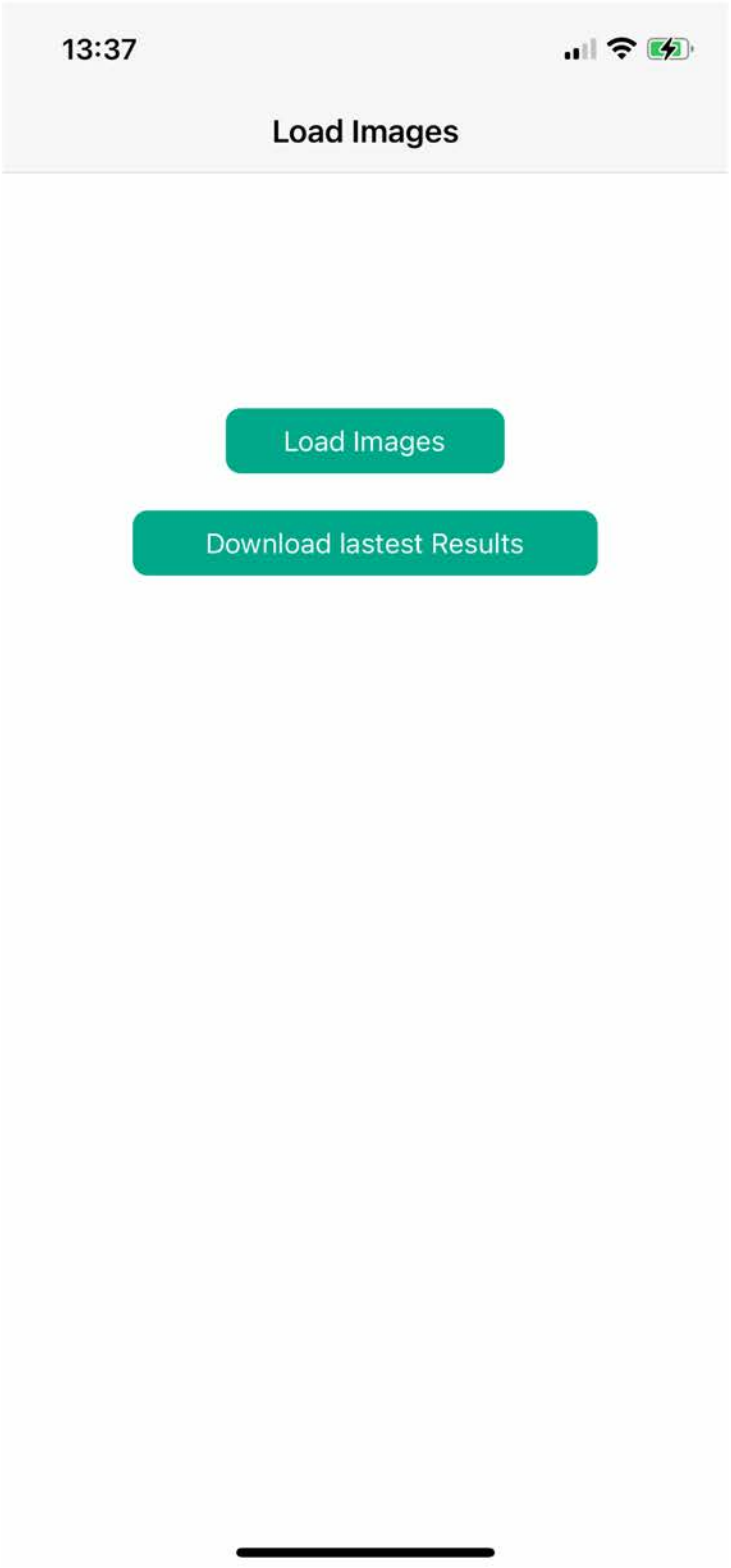



Figura 9: Pantalla inicial tras existir un entrenamiento previo.

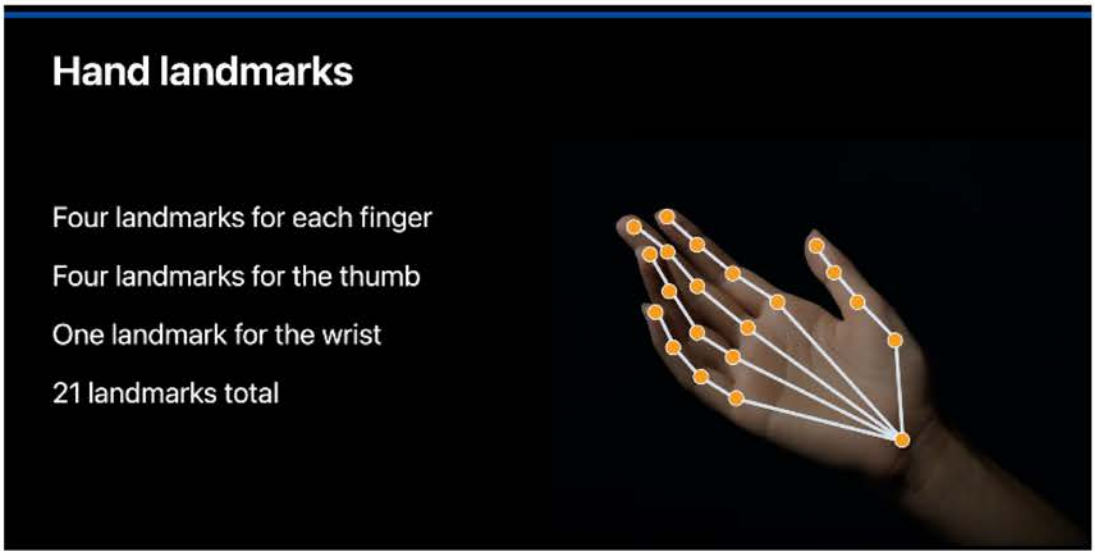



Figura 10: Hand Landmarks.

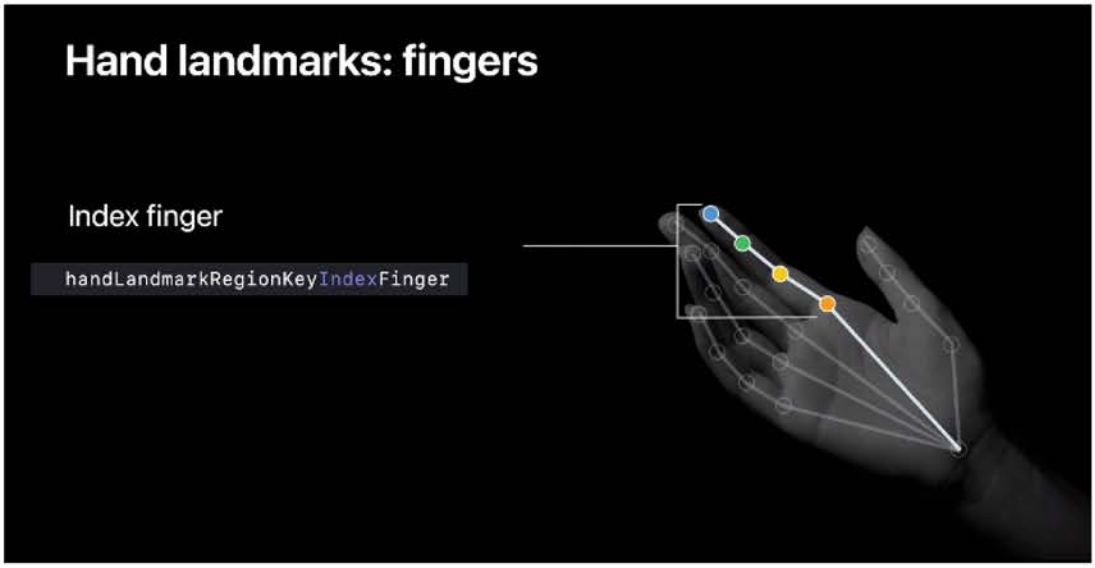



Figura 11: Index finger.

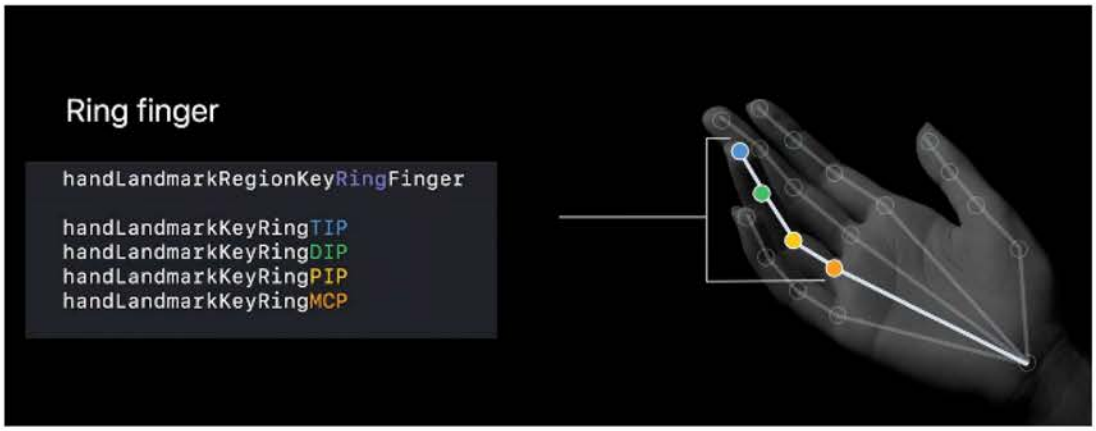



Figura 12: Ring finger.

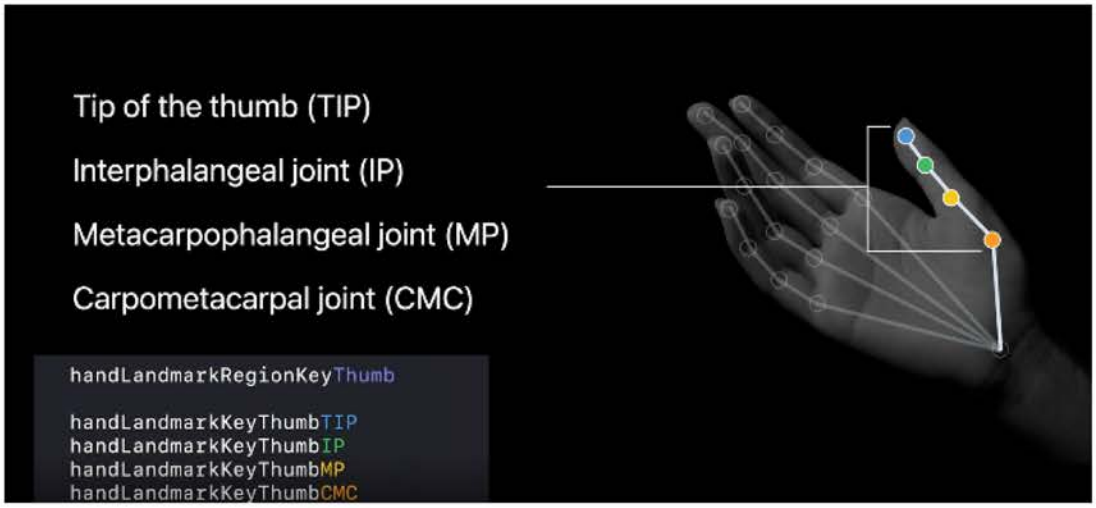



Figura 13: Thumb finger.

# 5 Modelo Deep Learning

## 5.1. Overview

En este sección se detalla el proceso de compilación, entrenamiento y posterior conversión a CoreML para su uso en un dispositivo iOS.

Se realiza bajo los siguientes componentes y herramientas:

- sistema operativo Ubuntu 20.04.1 LTS
- Entorno virtual Anacoda [3] para las instalación de herramientas a utilizar.
- Jupyter [21] como IDE para la programación del modelo de Deep Learning.

En el directorio con los datos del modelo, se incluye el fichero .yaml que incluye los paquetes necesarios para la configuración del entorno virtual a utilizar. De esta forma puede ser creado fácilmente el entorno virtual con la configuración deseada con la ejecución de este comando:

```
conda env create --file=kerasenv.yaml
```

El entorno utiliza los siguientes paquetes:

```
name: kerasenvTFGExtractedFeatures
channels:
  - defaults
dependencies:
  - python=3.6.8
  - pip
```

```
- scikit-learn
- scikit-image
- matplotlib
- seaborn
- ipython
- jupyter
- scipy
- numpy
- pandas
- pillow
- pydot
- keras=2.2.4
- tensorflow=1.14
- nomkl
- pip:
  - coremltools==3.0
```

Es especialmente importante las versiones utilizadas de TensorFlow [23] y Keras [11], puesto que la sintaxis del lenguaje cambia bastante entre versiones y no funcionará correctamente sin modificaciones en versiones posteriores.

## 5.2. Preparación de los datos de entrenamiento

La ejecución del notebook de Jupyter se encuentra en los apéndices del documento A, aquí se va a comentar los aspectos fundamentales de la construcción del modelo.

En el siguiente fragmento de código definimos las etiquetas que se van a interpretar, así como el directorio 'Dataset_short_v3', que contiene una carpeta train con las características obtenidas por la aplicación de entrenamiento.

Tras el procesado de datos, cada entrada de la variable X, es un vector de 42 valores representando las coordenadas de una letra. En el mismo índice de la variable Y, se encuentra la letra que representa esta entrada.

Para representar la letra, se utiliza un vector que utiliza one hot encoding. Esto significa que si estamos interpretando 24 letras, cada letra se representa como un vector de 24 posiciones,

donde el índice de la letra que representa tiene el valor 1, y el resto el valor 0. Se muestra un ejemplo de la representación de la letra A y la letra X en la figura 14

```
labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K",
        "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U",
        "V", "W", "X", "Y"]
num_classes = len(labels)

data_dir = "Dataset_short_v3"
train_dir = os.path.join(data_dir, "train")
batch_size = 32

# X: features list, Y: class list to predict
X = []
Y = []

# read each folder of train
for label in labels:
    name = os.path.join(train_dir, label, label + ".csv")
    df = pd.read_csv(name)
    X.extend(df.values)
    newSize = len(df)
    print("Letter %s: num_data: %d" %( label, newSize))
    for i in range(0,newSize):
        Y.append(label)

# convert to np array to have shape
X = np.array(X)
Y = np.array(Y)

#one hot encoding
lb = preprocessing.LabelBinarizer()
lb.fit(labels)
```

```
Y = lb.transform(Y)
```

```
In [17]: Y[0]

Out[17]: array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0])

In [19]: Y[12835]

Out[19]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 1])
```

Figura 14: One hot encoding de las letras A,X.

Se dividen los datos de entrada en tres sets: datos de entrenamiento, validación y test.

A cada letra se le establece un peso que determina la importancia de cada entrada de una letra en el entrenamiento del modelo. El dataset utilizado no está balanceado en el número de datos de entrada por cada letra. Esto se debe a que se ha realizado un proceso de selección en los datasets utilizados, y algunas letras tenían muy pocas imágenes válidas para el entrenamiento. A las clases con menos datos de entrada, se le da más importancia a cada entrada en el entrenamiento del modelo.

```
#split into training,val data. test_data
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.05)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.05)

#handle unbalance data
Y_train_classes = lb.inverse_transform(Y_train)
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(Y_train_classes),
                                                  Y_train_classes)
class_weight_dic = dict(enumerate(class_weights))
```

## 5.3. Compilación del modelo

La configuración del modelo será una red neuronal que contiene las siguientes capas:

1. Dense: Cada entrada es un vector de 42 valores y utiliza la función de activación 'relu'
2. Dense: Esta capa recibe como entrada la salida de la capa anterior, y está formada por 24 clases, una que representa a cada letra.
3. Activation: esta capa utiliza la función de activación 'softmax' y convierte cada entrada en una distribución de probabilidad en el que establece para cada tipo de letra, cual es la probabilidad de que la entrada inicial al modelo represente esa letra.

Los hiper parámetros han sido seleccionados tras realizar varias pruebas y son los siguientes:

- Función de optimización: Stochastic gradient descent.
- Learning rate: 0.01.
- Función de pérdida: Stochastic gradient descent

```
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(42,)))
model.add(Dense(num_classes))
model.add(Activation("softmax"))

model.compile(optimizer=optimizers.SGD(lr=0.01), loss="Stochastic gradient
    descent",
                  metrics=["accuracy"])
```

Esto produce un modelo con un total de 2,168 parámetros que pueden ser entrenados como se muestra en la figura 15, y la estructura de capas que se muestra en la figura 16

## 5.4. Entrenamiento

Se procede al entrenamiento del modelo hasta que se produzcan varias iteraciones consecutivas en el que su porcentaje de acierto en el dataset de validación no mejora. Cuando esto

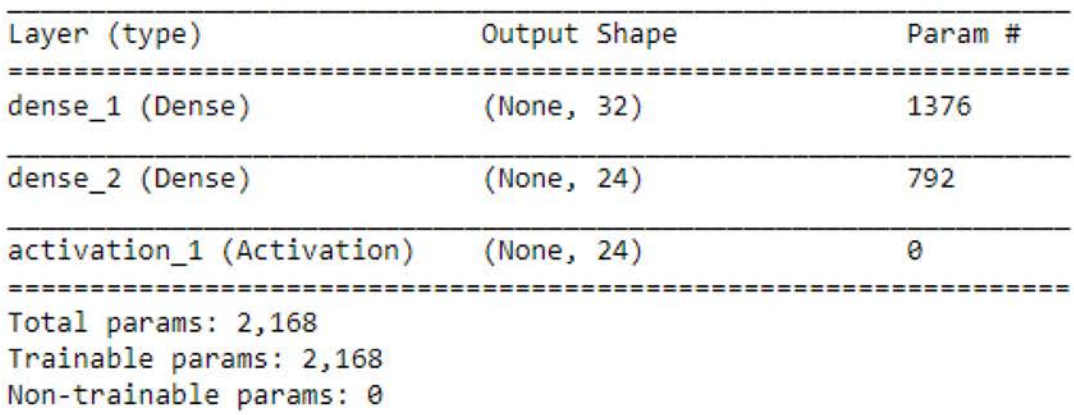



Figura 15: Resumen del modelo.

ocurra, procedemos a bajar el hiper parámetro Learning Rate a la mitad y volvemos a realizar el entrenamiento.

```
histories.append(model.fit(
    X_train,
    Y_train,
    batch_size=batch_size,
    epochs=200,
    validation_data=(X_val, Y_val),
    callbacks=my_callbacks,
    class_weight=class_weight_dic))

#finetuning
K.set_value(model.optimizer.lr,
                K.get_value(model.optimizer.lr) / 2)
```

Este proceso varía de una ejecución a otra porque los pesos iniciales del modelo son elegidos arbitrariamente, en este ejemplo se consiguió un porcentaje de acierto en el dataset de validación de 94.754 % 17

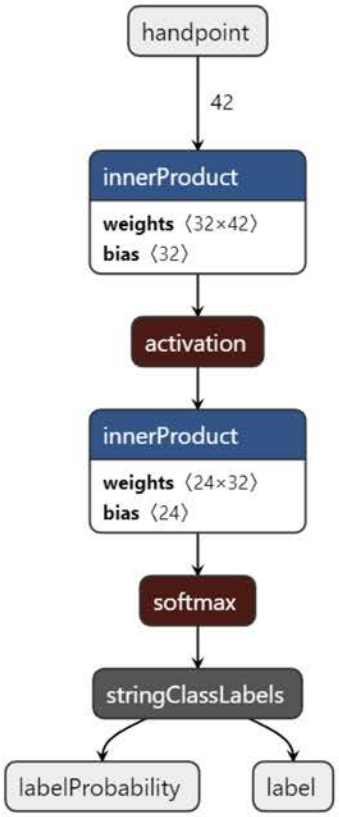



Figura 16: Capas del modelo.

## 5.5. Convertir a CoreML

Como último paso, tendremos que convertir el modelo a un formato compatible con iOS: Esto se realiza con CoreMLTools [10] Cargamos el modelo con el mejor resultado del entrenamiento, y mediante esta herramienta lo convertimos a CoreML [9].

El modelo tendrá una entrada llamada 'handpoint' que será un vector de 42 valores, y tendrá dos salidas:

- labelProbability: Distribución de probabilidad de 24 valores, en el que cada valor representa la probabilidad de que la entrada represente esa letra.
- label: Letra con la mayor probabilidad de acierto.

```
Epoch 00029: val_acc did not improve from 0.94754
Epoch 30/200
11584/11584 [==============================] - 0s 30us/step - loss: 0.2251 - acc: 0.9505 - val_loss: 0.2344 -

Epoch 00030: val_acc did not improve from 0.94754
Epoch 31/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2245 - acc: 0.9507 - val_loss: 0.2315 -

Epoch 00031: val_acc did not improve from 0.94754
Epoch 32/200
11584/11584 [==============================] - 0s 32us/step - loss: 0.2238 - acc: 0.9512 - val_loss: 0.2336 -

Epoch 00032: val_acc did not improve from 0.94754
Epoch 33/200
11584/11584 [==============================] - 0s 29us/step - loss: 0.2226 - acc: 0.9517 - val_loss: 0.2321 -

Epoch 00033: val_acc did not improve from 0.94754
Epoch 00033: early stopping
```

Figura 17: Resultados del entrenamiento del modelo.

```
import coremltools
from keras.models import load_model
best_model = load_model(checkpoint_dir + "featuresTFGNotebook-0.2425-0.9475.hdf5")
coreml_model = coremltools.converters.keras.convert(
        best_model,
        input_names="handpoint",
        output_names="labelProbability",
        predicted_feature_name="label",
        class_labels=labels)

# add metadata to the model
coreml_model.author = "Daniel Gallego Peralta"
coreml_model.license = "Public"
coreml_model.short_description = "Hand points classifier for 24 different letters
    of ASL"

coreml_model.input_description["handpoint"] = "normalized coordinates for hand
    points"
coreml_model.output_description["labelProbability"]= "Prediction probabilities"
coreml_model.output_description["label"]= "Class label of top prediction"
```

```
coreml_model.save("ASLHandPointTFG.mlmodel")
```

Este archivo 'ASLHandPointTFG.mlmodel' se genera en el directorio actual del notebook y es el modelo en formato CoreML de Deep Learning que utiliza la aplicación iOS.

# 6

# ASL Interpreter

## 6.1. Overview

Se desarrolla una aplicación móvil para iOS cuyo objetivo es interpretar el alfabeto de la lengua de signos americana. Permitirá seleccionar videos mediante diferentes métodos de entrada o la cámara del dispositivo, realizará predicciones sobre los datos de entrada y mostrará por pantalla los resultados. La aplicación está desarrollada únicamente para iPhone y ha sido probada con un iPhone 12 Pro. Es posible que funcione en dispositivos de menor potencia pero no está entre los objetivos del proyecto.

Landscape right es la única orientación soportada por el proyecto, además para obtener los mejores resultados, es recomendable que el dispositivo esté sujeto con un trípode para mejorar su estabilidad. La configuración del proyecto se puede ver en la figura 18.

## 6.2. Arquitectura

La aplicación se desarrolla en Swift 5 y siguiendo la arquitectura Clean Swift Architecture [8]. Los objetivos de esta arquitectura son los siguientes:

1. Disminuir la importancia del controlador: Habitualmente en la arquitectura más comúnmente usada en el desarrollo de aplicaciones para iOS, Model View Controller, el controlador acaba realizando la mayor parte del trabajo. En esta arquitectura, el controlador pasará a ser parte de la vista y el Interactor será el encargado de realizar la parte lógica de la aplicación, pudiendo utilizar diferentes Workers para ello.

2. Seguir un flujo de comunicación unidireccional: La comunicación entre los componentes siguen siempre la misma dirección 19

Siguiendo esta arquitectura, la aplicación cuenta con las siguientes escenas:

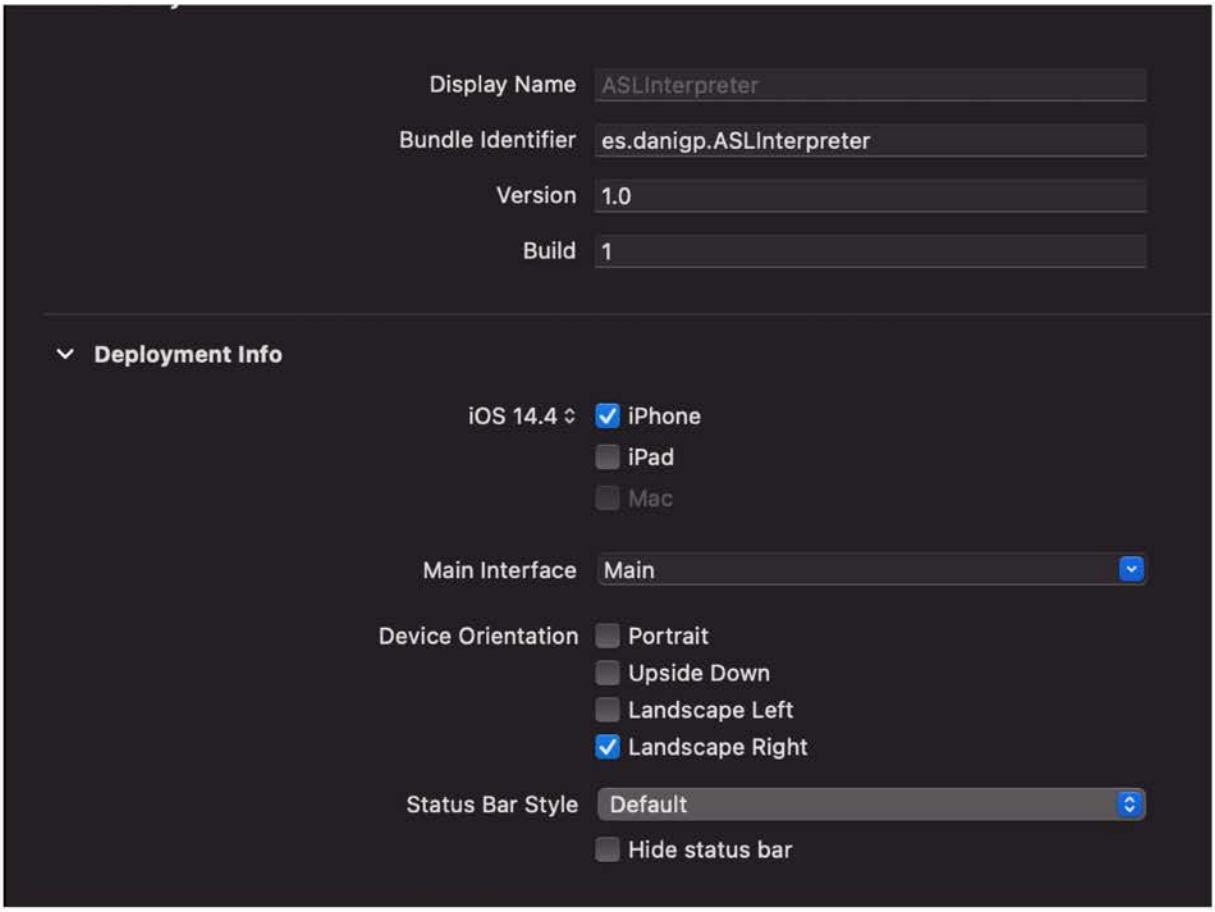



Figura 18: Configuración del proyecto en Xcode.

- Source picker: Encargada de escoger el método de entrada, puede ser tanto la cámara del dispositivo como vídeos. Los videos pueden estar localizados tanto en el propio dispositivo como en iCloud.

- Configuración: Permite seleccionar diferentes configuraciones de funcionamiento. Se almacenan de forma persistente para ser usadas entre diferentes ejecuciones.

- Overlay: Escena encargada de coordinar los datos de entrada con el intérprete de ASL. Esta escena no es visible pero su función es presentar y coordinar las siguientes escenas.

- Cámara: Se encarga de proporcionar un buffer de imágenes para su uso por el intérprete. Obtiene los frames desde la cámara del dispositivo o desde un fichero de video y los transmite al intérprete.

- Intérprete: Recibe un buffer de imágenes y realiza predicciones sobre el modelo de Deep

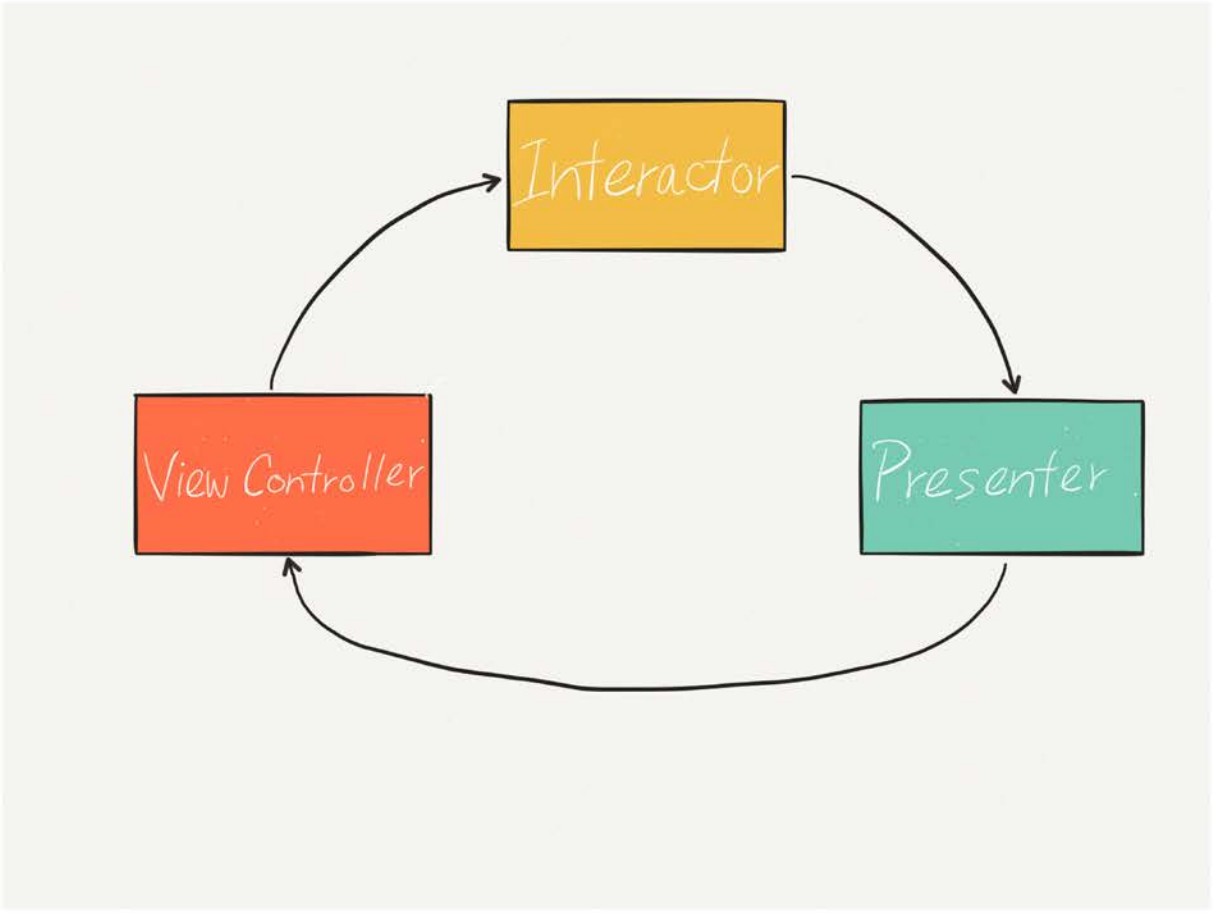



Figura 19: Clean Swift Cycle

Learning. Tras un proceso de refinamiento los resultados se muestran por pantalla.

## 6.3. Source Picker

Es la escena inicial de la aplicación. Permite seleccionar cuál va a ser la fuente de datos. La interfaz se puede ver en la figura 20, y cuenta con las siguientes opciones:

- Live Camera: Se selecciona utilizar la cámara del dispositivo. La escena no necesita realizar ninguna acción en este caso.
- Library: Permitirá seleccionar un video desde la librería de fotos de iOS. Para ello, se utilizará la clase UIImagePickerViewController que se encarga de mostrar la interfaz de selección de videos y devolver un fichero de video en caso de que sea seleccionado. La interfaz se puede ver en la figura 21
- iCloud: Permitirá seleccionar vídeos desde iCloud. Se utiliza la clase UIDocumentPickerViewController que se encarga de mostrar la interfaz de selección de videos y devolver un fichero de video en caso de que sea seleccionado. La interfaz se puede ver en la figura 22
- Configuración: Permite dirigirse a la escena de configuración de la aplicación.

Si se ha seleccionado una entrada de datos, se dirige a la escena Overlay para su procesamiento.

## 6.4. Camera

Es la escena encargada de proporcionar un buffer de imágenes para su procesamiento, que puede ser obtenido desde la cámara del dispositivo o desde un fichero de video. Por lo tanto, se distinguen dos modos de funcionamiento:

- Cámara del dispositivo: se utiliza la cámara posterior del dispositivo para obtener la mayor calidad de imágen. Se intenta obtener una resolución de 1920x1080. La entrada se mostrará por pantalla utilizando la siguiente clase AVCaptureVideoPreviewLayer. El procesamiento se realizará en segundo plano utilizando una cola para ello.

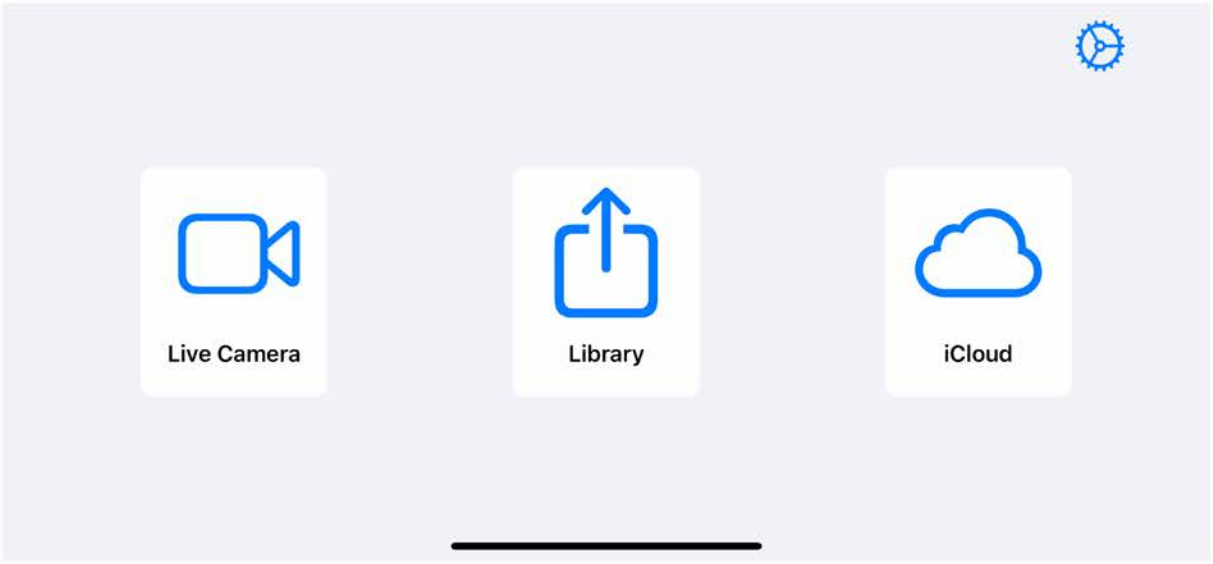



Figura 20: Interfaz de la escena Picker View.

Se muestran los fragmentos más importante para su configuración:

Queue

```
    private let cameraQueue = DispatchQueue(label: "CameraDataOutput",
                                            qos: .userInitiated, attributes:
[],
                                            autoreleaseFrequency: .workItem)
```

Configuración

```
        let wideAngle = AVCaptureDevice.DeviceType.builtInWideAngleCamera
        let discoverySession = AVCaptureDevice.DiscoverySession(deviceTypes:
[wideAngle],
                                                                mediaType:
.video, position: .unspecified)

        let session = AVCaptureSession()
        session.beginConfiguration()
        session.sessionPreset =
videoDevice.supportsSessionPreset(.hd1920x1080) ? .hd1920x1080 : .high
```

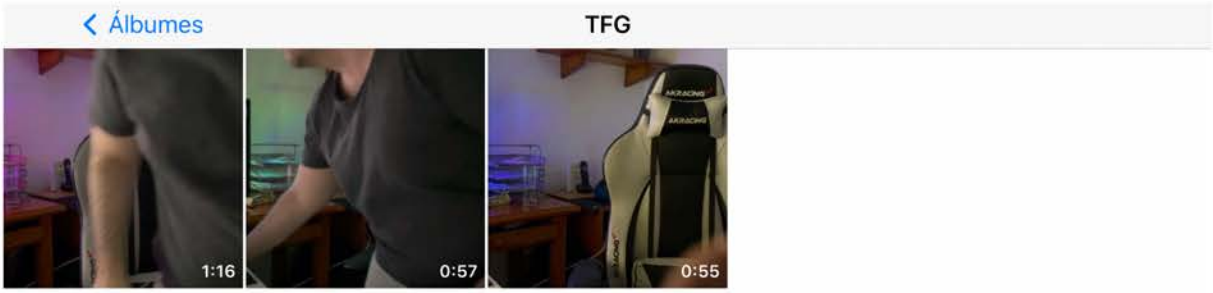



Figura 21: Interfaz de selección de vídeos del dispositivo.

```
    let dataOutput = AVCaptureVideoDataOutput()
    if session.canAddOutput(dataOutput) {
        session.addOutput(dataOutput)
        dataOutput.alwaysDiscardsLateVideoFrames = true
        dataOutput.videoSettings = [
            String(kCVPixelBufferPixelFormatTypeKey):
Int(kCVPixelFormatType_420YpCbCr8BiPlanarFullRange)
        ]
        dataOutput.setSampleBufferDelegate(self, queue: cameraQueue)
    } else {
        showError("can not add output")
    }
```

- Vídeo: Se obtiene un fichero de vídeo encapsulado en un objeto de tipo AVAsset y se utiliza un objeto de la clase AVPlayer para mostrarlo por pantalla. El video será reproducido inmediatamente tras ser mostrado en pantalla. El procesamiento se realizará en segundo plano utilizando una cola para ello.

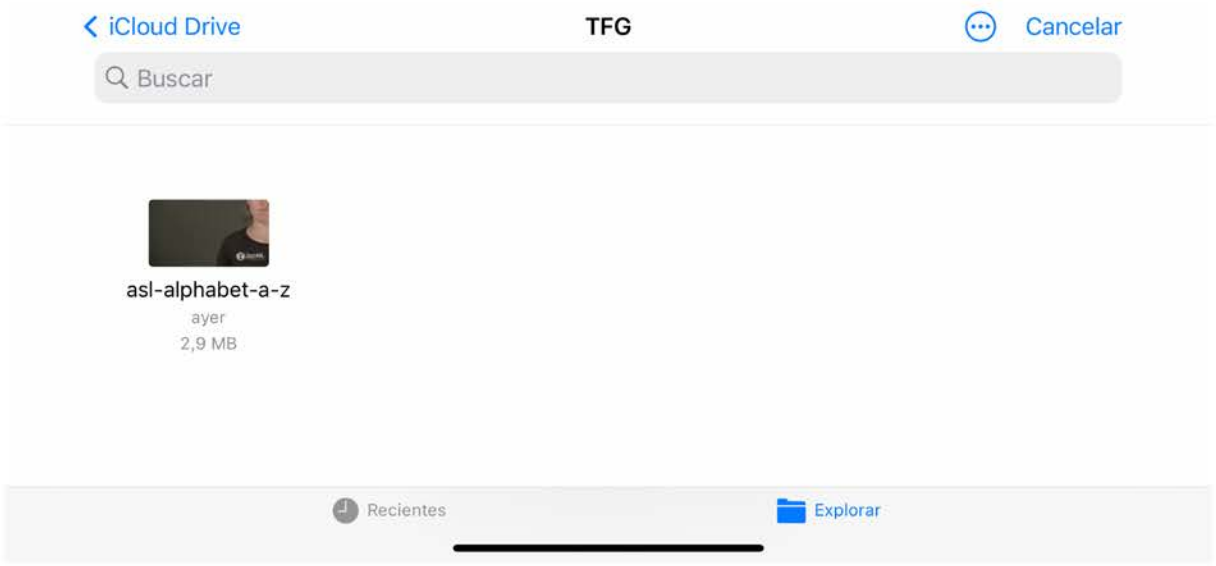



Figura 22: Interfaz de selección de vídeos desde iCloud.

Se muestran los fragmentos más importante para su configuración:

```
videoFileQueue.async {

        // Create sample buffer from pixel buffer
        var sampleBuffer: CMSampleBuffer?
        var formatDescription: CMVideoFormatDescription?
        CMVideoFormatDescriptionCreateForImageBuffer(allocator: nil,
imageBuffer: pixelBuffer, formatDescriptionOut: &formatDescription)
        let duration = self.videoFileFrameDuration
        var timingInfo = CMSampleTimingInfo(duration: duration,
presentationTimeStamp: itemTime, decodeTimeStamp: itemTime)
        CMSampleBufferCreateForImageBuffer(allocator: nil,
                                           imageBuffer: pixelBuffer,
                                           dataReady: true,
                                           makeDataReadyCallback: nil,
                                           refcon: nil,
                                           formatDescription:
formatDescription!,
                                           sampleTiming: &timingInfo,
                                           sampleBufferOut:
```

```
&sampleBuffer)
        if let sampleBuffer = sampleBuffer {
            self.delegate?.cameraViewController(self, didReceiveBuffer:
sampleBuffer, orientation: self.videoFileBufferOrientation)
        }
    }
}
```

Esta clase actúa como delegado del intérprete y se encarga de transmitir un buffer de imágenes. Para ello, se utiliza el siguiente protocolo:

```
protocol CameraViewControllerDelegate: class {
    func cameraViewController(_ controller: CameraViewController,
didReceiveBuffer buffer: CMSampleBuffer, orientation:
CGImagePropertyOrientation)
}
```

## 6.5. Interpreter ASL

Es la escena principal de la aplicación y del proyecto. Se encarga de realizar predicciones utilizando como entrada un buffer de imágenes y mostrando por pantalla los resultados. Se compone de los siguientes modos de funcionamiento que pueden configurarse en la escena de configuración:

- Free : Muestra por pantalla los resultados obtenidos de las predicciones sin ningún procesamiento posterior. Es una vista en tiempo real de las predicciones devueltas por el modelo de Deep Learning.
- Letter occurrences: Tras obtener una predicción, se realiza un proceso de refinamiento para obtener resultados más fiables. Para confirmar una predicción como correcta, se requieren un mínimo número de predicciones consecutivas con el mismo resultado. Puede

configurarse obtener varias predicciones consecutivas iguales o un mínimo de ocurrencias sobre un total. Por ejemplo, 5 de 6 predicciones de la misma letra para confirmarse como correcta. Estos parámetros "Minimum letters to compare resultsz "Minimum occurrences of letters detected"pueden ser configurados en la escena de configuración.

- Detect words: Se encarga de ir formando palabras con las letras detectadas. También utiliza el proceso de refinamiento comentado anteriormente para mejorar los resultados. A medida que se detectan letras se muestran por pantalla formando una palabra hasta que se decide que la palabra ha terminado. Para interpretar cuando una palabra empieza y acaba, se necesita un procesamiento adicional. Mediante un gesto con la mano, podremos indicar cuando una palabra empieza y acaba.

La interfaz cuenta con los siguientes elementos que pueden visualizarse en la figura 23:

- En la esquina inferior izquierda se muestran las 3 predicciones con la mayor probabilidad de ser correcta. Estos son los resultados devueltos por el modelo de Deep Learning en tiempo real sin ningún procesamiento. Estos resultados son sólo visibles en modo depuración, esto puede establecerse en la escena de configuración.

- En la esquina inferior derecha se muestra la letra con la probabilidad más alta detectada en el caso de que se haya confirmado como correcta. Según el modo de funcionamiento se pueden necesitar varias predicciones consecutivas de la misma letra para confirmarse. En caso de que no se confirme ninguna letra esta vista no será visible o contendrá los últimos resultados válidos.

- Bounding Box: Rectángulo para visualizar la mano que se ha utilizado para la predicción. En la escena de configuración podemos elegir si se utiliza la mano izquierda o derecha.

El código completo es bastante extenso así que se van a mostrar las configuraciones más importantes.

El primer paso es detectar la mano es una imagen, para ello seguiremos el mismo procedimiento que se utilizó en la aplicación de entrenamiento para la extracción de características. Utilizamos Vision Framework, y mediante la petición VNDetectHumanHandPoseRequest se obtiene la detección de hasta dos manos en una imagen.

```
override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        handPoseRequest = VNDetectHumanHandPoseRequest()
        handPoseRequest.maximumHandCount = 2
}

 func checkObservations(_ controller: CameraViewController, buffer:
    CMSampleBuffer, orientation: CGImagePropertyOrientation) throws {
    let visionHandler = VNImageRequestHandler(cmSampleBuffer: buffer, orientation:
    orientation, options: [:])
    try visionHandler.perform([handPoseRequest])
```

Se utiliza el mismo modelo y proceso para la obtención y normalización de coordenadas que la aplicación de entrenamiento. El modelo Hand es el mismo en ambas aplicaciones.

```
func processObservations(_ observation: VNHumanHandPoseObservation) -> Hand? {
    guard observation.confidence ≥ minConfidenceObservation else {
        print("observation low confidence")
        return nil
    }

    do {

        let wrist = try observation.recognizedPoint(.wrist)
        let indexFingerPoints = try observation.recognizedPoints(.indexFinger)
        let ringFingerPoints = try observation.recognizedPoints(.ringFinger)
        let thumbFingerPoints = try observation.recognizedPoints(.thumb)
        let littleFingerPoints = try observation.recognizedPoints(.littleFinger)
        let middleFingerPoints = try observation.recognizedPoints(.middleFinger)

        let hand = HandFingerPoints(indexFingerPoints: indexFingerPoints,
                                    ringFingerPoints: ringFingerPoints,
                                    littleFingerPoints: littleFingerPoints,
```

```
                                middleFingerPoints: middleFingerPoints,
                                thumbFingerPoints: thumbFingerPoints,
                                wristPoint: wrist)

            hand.normalizeHand()
            numberOfProcess += 1
            return hand
        } catch {
            print(error)
            return nil
        }
}
```

Sobre el modelo obtenido, realizamos la extracción de características que consiste en obtener una array de 42 valores de tipo Double que representan la posición relativa de diferentes posiciones de la mano. El método extractIfValidFeatures recibe un parámetro que indica si debemos invertir el valor de las coordenadas del eje X.

EL modelo ha sido entrenado en su totalidad con fotos de la mano derecha de diferentes personas. Para mejorar los resultados es importante que tanto los datos de entrenamiento como los datos de entrada para realizar una predicción sean sobre la misma mano. Sin embargo, podemos realizar también predicciones con la mano izquierda con tan sólo invertir los valores del eje X. Todas las coordenadas están normalizadas de 0.0 a 1.0, así que para invertir el valor de una posición simplemente restamos a 1.0 el valor correspondiente. La mano utilizada para realizar la predicción puede ser configurada en la escena de configuración.

Como respuesta del método 'prediction(multiArrayBuffer: multiArrayBuffer)', obtenemos una distribución de probabilidad. Se filtran las 3 predicciones con mayor probabilidad y se muestran por pantalla. También se obtiene el mejor resultado para su procesamiento. El Interactor será el encargado de recibir la letra detectada y volverá a comunicarse con el Intérprete para transmitir el resultado, en caso de que la predicción se confirme como correcta.

```
guard let hand =
    VisionProcessImage.sharedInstance.processObservations(observation),
    let features = hand.extractIfValidFeatures(flipX:
```

```
    ASLConfiguration.shared.handCase == ASLConfiguration.HandCase.left) else {
    print("failed to extract hand features")
    return
}
if let multiArrayBuffer = try? MLMultiArray(features) {
    let labelProbability = ASLConfiguration.shared.prediction(multiArrayBuffer:
    multiArrayBuffer)
    let top3 = labelProbability.sorted {
        return $0.value > $1.value
    }.prefix(3)

    let descriptionTop3 = top3.map { observation in
        String(format: "%@ %.1f%%", observation.key, observation.value * 100)}
    DispatchQueue.main.async {
        if ASLConfiguration.shared.isDebugMode {
            self.aslView.labelResults.text = descriptionTop3.joined(separator:
    "\n")
        }
    }

    if let betterPrediction = top3.first, betterPrediction.value >
    Double(ASLConfiguration.shared.letterDetectionMinConfidence) {
        interactor?.topPredictionLetter(betterPrediction.key)
    }
}
```

La clase ASLInterpreterWorker es la encargada de confirmar una predicción como correcta o no. Esto dependerá del modo de funcionamiento de la aplicación así como de los diferentes parámetros establecidos. En el modo de funcionamiento Free 6.5, siempre devolverá una predicción como correcta. En el modo de funcionamiento Letter Occurrences 6.5, se trata de confirmar si de las últimas X predicciones, al menos Y son de la misma letra. En este caso, esta letra se confirma como correcta.

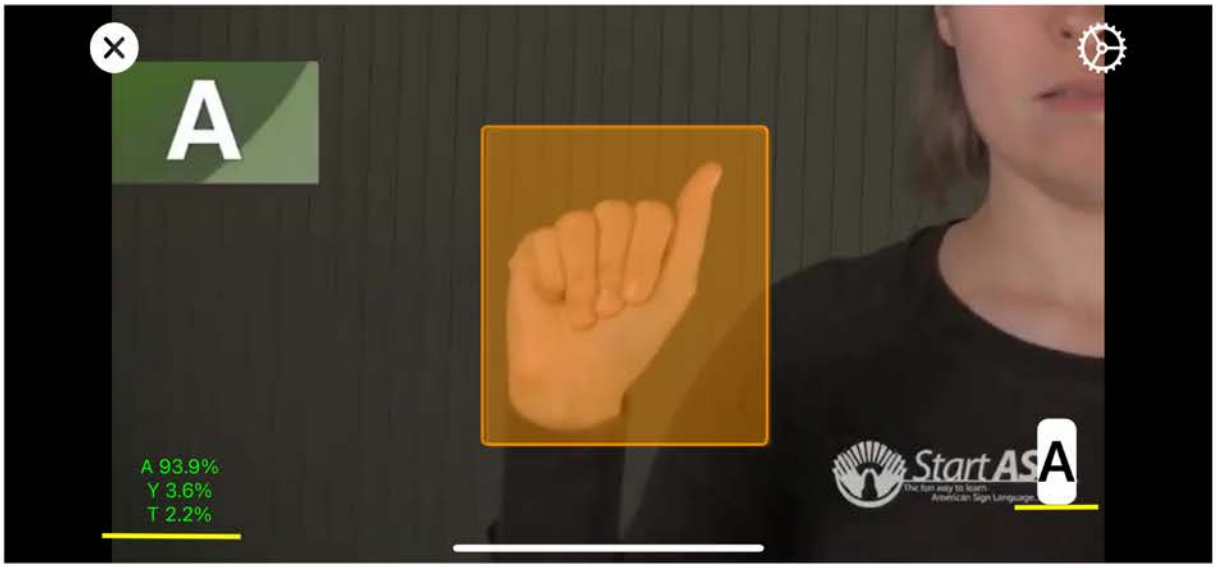



Figura 23: Interfaz intérprete ASL.

```
func detectMinOcurrencesLetter(_ letter: String) {
    queue.async { [unowned self] in
        self.predictions.append(letter)

        if predictions.count ≥ ASLConfiguration.shared.minLettersToCompare {
            let slide =
    predictions.suffix(ASLConfiguration.shared.minLettersToCompare)
            let lastPredictions = Array(slide)

            let occurences: [Occurrence] = getNumberOfOccurrences(lastLetters:
    lastPredictions)
            guard let topResult = getTopResult(ocurrences: occurences) else {
                return
            }

            if topResult.times ≥ ASLConfiguration.shared.minOcurrences &&
    topResult.letter ≠ lastLetter {
                lastLetter = topResult.letter
                predictions.removeAll()
                delegate?.aslInterpreter(self, didDetectWord: topResult.letter)
```

```
            }
        }
    }
}
```

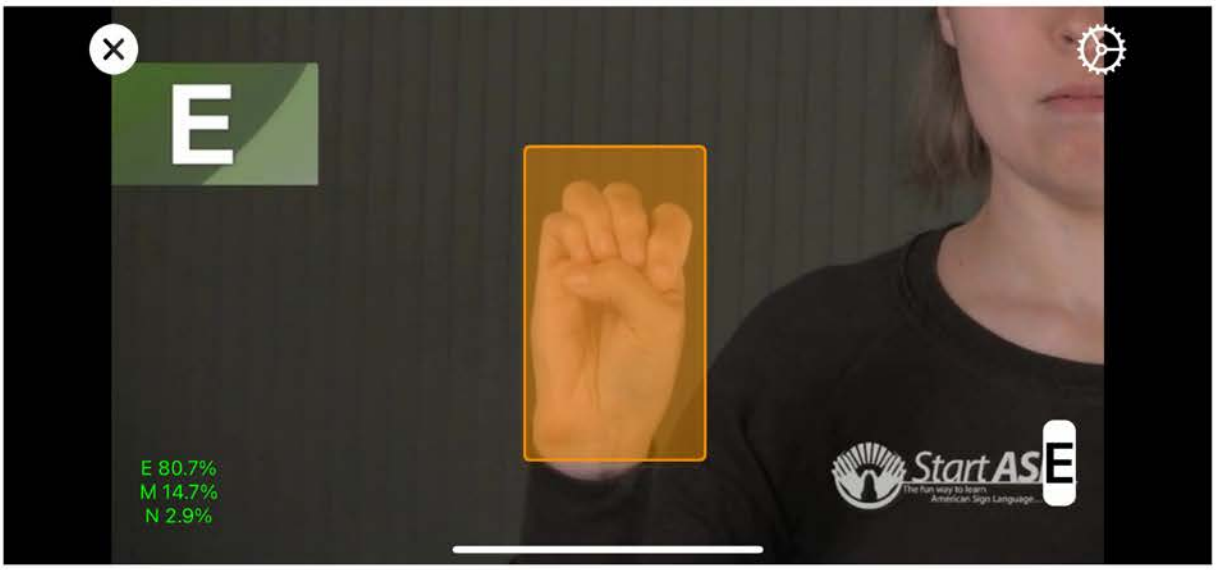



Figura 24: Interfaz intérprete ASL.


Figura 25: Interfaz intérprete ASL.

Para el modo de funcionamiento Detect Words 6.5, se necesita configuración adicional. Necesitamos detectar cuándo empieza y termina una palabra. Para ello, vamos a utilizar un

detector de gestos. El procedimiento es el siguiente:

Se utilizará una mano para las representación de las letras, y la otra mano para realizar un gesto. Por tanto ambas manos necesitan estar visibles para este modo de funcionamiento. En caso de seleccionar en la escena de configuración la mano derecha, se utilizará la mano izquierda para interpretar el gesto.

El gesto utilizado para indicar cuando empieza y termina una palabra es denominado Pinch. Consiste en tener los dedos índices y pulgar juntos. Se detecta la posición de los dedos índices y pulgar, y dependiendo si la distancia entre ellos es inferior a un valor mínimo, se considera que se encuentra en un estado Pinch 26, en el caso contrario, el estado es apart 27.

```
guard let middlePoints = try? rightHand.recognizedPoints(.thumb),
      let indexFingerPoints = try? rightHand.recognizedPoints(.indexFinger),
      let middleTipPoint = middlePoints[.thumbTip],
      let indexTipPoint = indexFingerPoints[.indexTip] else {
    return
}

let minConfidenceFinger: VNConfidence = 0.3
guard middleTipPoint.confidence > minConfidenceFinger && indexTipPoint.confidence
    > minConfidenceFinger else {
    return
}

let thumbTip = middleTipPoint.location.applying(.verticalFlip)
let indexTip = indexTipPoint.location.applying(.verticalFlip)

let thumbTipLayer = controller.viewPointForVisionPoint(thumbTip)
let indexTipLayer = controller.viewPointForVisionPoint(indexTip)
gestureProcessor.newPoints(thumbTip: thumbTipLayer, indexTip: indexTipLayer)
```

La clase HandGestureProcessor es la encargada de ir recibiendo estos dos puntos y determinar en el estado se encuentra:

1. possiblePinch: Se han detectado evidencias de que se ha realizado un gesto pinch, pero

no se han alcanzado en mínimo número de evidencias para confirmarlo.

2. pinched: se ha confirmado que se encuentra en el estado pinched.

3. possibleApart: partiendo del estado pinched, se ha detectado alguna evidencia de que los dedos no están lo suficientemente juntos, pero aún no esta confirmado.

4. apart: Se ha realizado la transición del estado pinched al estado apart, se detecta como el fin de una palabra.

5. unknown: No determinado. Si pasan más de dos segundos sin recibir ninguna información o aún no se ha recibido ninguna información, se encuentra en este estado.

Definición de estados:

```
enum State {
    case possiblePinch
    case pinched
    case possibleApart
    case apart
    case unknown
}
```

Según una distancia mínima establecida pinchMaxDistance, se modifica el estado actual.

```
func newPoints(thumbTip: CGPoint, indexTip: CGPoint) {
    lastEventDate = Date()
    let distance = indexTip.distance(from: thumbTip)
    if distance < pinchMaxDistance {
        pinchEvidenceCounter += 1
        apartEvidenceCounter = 0
        state = (pinchEvidenceCounter ≥ evidenceCounterStateTrigger) ? .pinched :
    .possiblePinch
    } else {
        apartEvidenceCounter += 1
        pinchEvidenceCounter = 0
```

Figura 26: Estado pinched.

```
        state = (apartEvidenceCounter ≥ evidenceCounterStateTrigger) ? .apart :
    .possibleApart
    }
}
```

La aplicación sólo realiza predicciones sobre el modelo de Deep Learning si se encuentra entre los estados possiblePinch, pinched o possibleApart. En caso contrario, únicamente se intenta identificar si se ha realizado un gesto Pinch.

Cuando se confirma el estado apart, se confirma el fin de una palabra. Al establecer el fin de una palabra, en la siguiente predicción, se eliminaran los resultados anteriores de la pantalla y se empieza una palabra sin ninguna letra anterior.

```
if ASLConfiguration.shared.isTextCheckerEnabled {
    print("textChecking")
```

Figura 27: Estado apart.

```
    var newWord = word.joined()
    let language = "en"
    let rangeMisspelled = textChecker.rangeOfMisspelledWord(in: newWord, range:
    NSRange(0..<newWord.utf16.count), startingAt: 0, wrap: false, language:
    language)
    if rangeMisspelled.location != NSNotFound {
        if let guess = textChecker.guesses(forWordRange: rangeMisspelled, in:
    newWord, language: language)?.first {
            newWord = guess
            print("newWord: guess: \(guess)")
        }
    }
    delegate?.aslInterpreter(self, didDetectWord: newWord)
}
```

## 6.6. Configuración

Por último, tenemos una escena de configuración, a la cual se puede acceder desde cualquier escena y que permite cambiar diferentes parámetros que son utilizados por la aplicación. Se puede pausar la ejecución de cualquier video o cámara, cambiar los parámetros y continuar por el mismo punto con los nuevos parámetros actualizados. La interfaz se puede ver en la figura 28 y 29.

Los parámetros son los siguientes:

1. Minimal hand confidence: Es usado para la detección de una mano en una imagen. Se utiliza para filtrar los resultados obtenidos por Vision Framework [4]. Las predicciones con un nivel de confianza menor al establecido son descartadas.

2. Minimal letter confidence: Es usado para la detección de letras sobre el modelo de Deep Learning. Cualquier predicción con un nivel de confianza inferior es descartado.

3. Mode: Define el modo de funcionamiento de la aplicación. Los modos son definidos en el punto 6.5.

4. Minimum letters to compare results: Se utiliza para el modo de funcionamiento Letter occurrences y Detect Words. Determina el mínimo número de predicciones necesarias antes de poder confirmar una predicción como correcta.

5. Minimum occurrences of letters detected: Se utiliza para el modo de funcionamiento Letter occurrences y Detect Words. Determina el mínimo número de ocurrencias de una misma letra sobre las últimas predicciones para confirmar la letra como correcta.

6. Filter hand by mid X coordinate: En caso de estar habilitado, si estamos realizando predicciones sobre la mano derecha, la mano necesita estar localizada en la primera mitad de la pantalla sobre el eje horizontal, en caso contrario es rechazada. Si es con la mano izquierda es en sentido inverso. Esto sirve para rechazar predicciones erróneas en las transiciones entre letras. Para un buen funcionamiento bajo este modo, es necesario que la persona que realiza los gestos esté centrada en la imagen.

7. Hand: Permite seleccionar la mano sobre la que se realizarán predicciones. En caso de

solo ser visible una mano, se utiliza sin importar cual esté seleccionada en este parámetro.

8. Modelo: Parámetro utilizado para el desarrollo del proyecto. Permite seleccionar entre diferentes modelos de Deep Learning que se han ido desarrollando en diferentes iteraciones del proyecto. El modelo seleccionado por defecto es el que tiene los mejores resultados con un porcentaje del 95 % de precisión en el test de validación.

9. Debug: En caso de estar habilitado, imprime diferentes mensajes por pantalla para ayudar en el desarrollo. Así como habilita la vista de resultados en tiempo real, esta vista se puede ver en la figura 25.

10. Resets to defaults: Restaurar todos los parámetros a su configuración inicial.

La configuración inicial es la que ha dado mejores resultados, especialmente para la confirmación de predicciones, obtener 6 predicciones consecutivas de la misma letra es un buen indicador de que es una buena predicción. Esto puede provocar que alguna vez se rechace alguna predicción que es correcta, pero devuelve los mejores resultados para tener un buen balance y evitar predicciones erróneas que pueden producirse entre transiciones entre letras o por pequeñas variaciones en la mano.

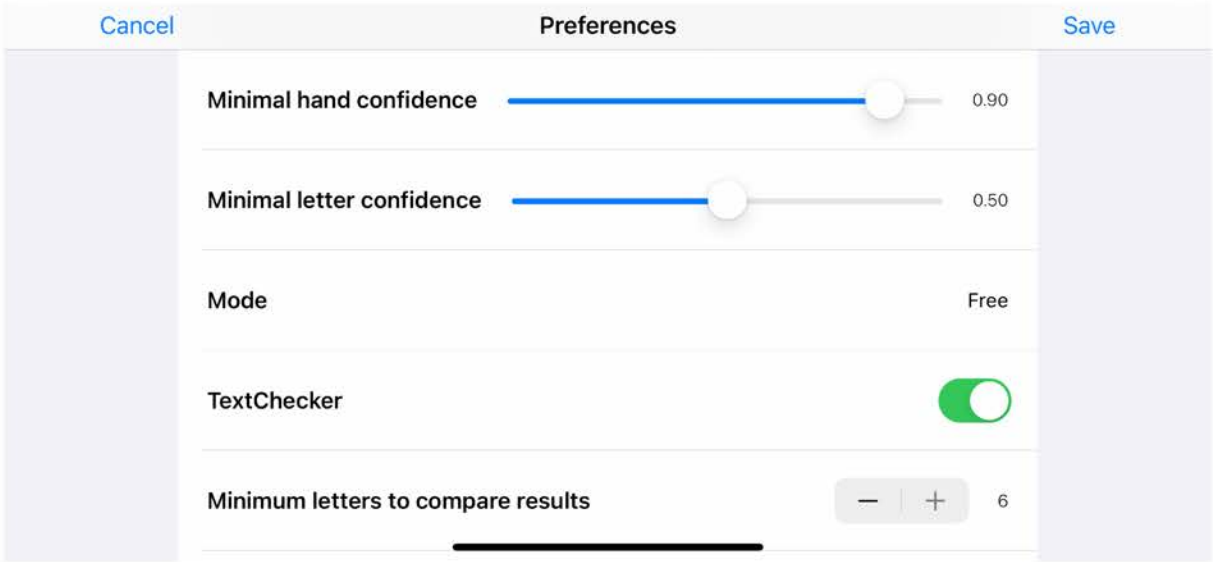



Figura 28: Parámetros de configuración.

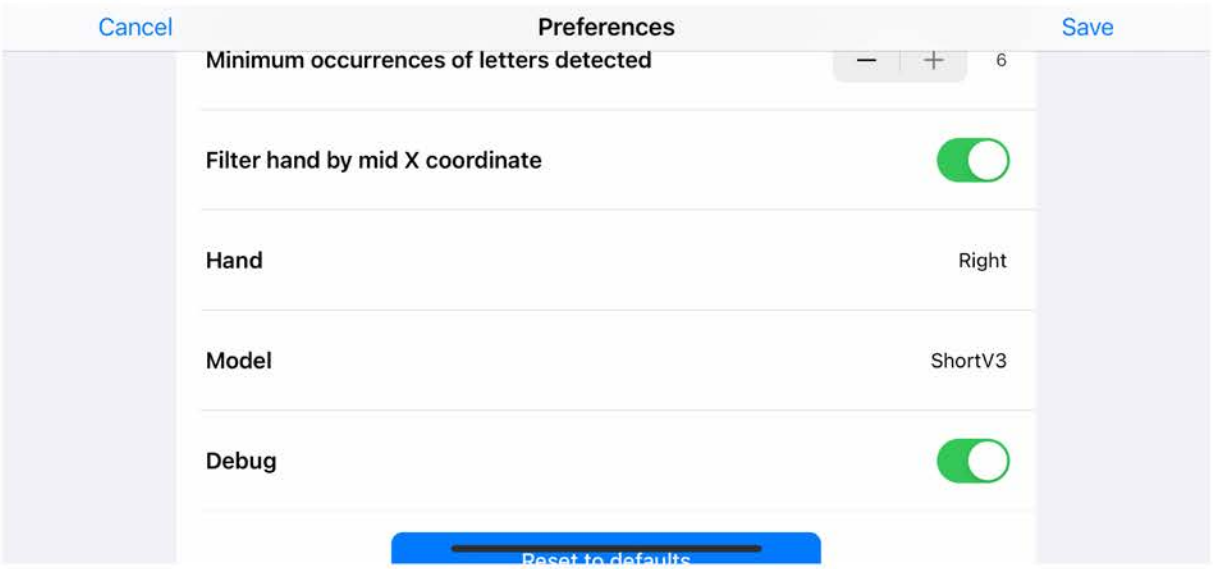



Figura 29: Parámetros de configuración.

# 7

# Resultados y pruebas

## 7.1. Modelo Deep Learning

Como se puede ver en el apéndice A, en el entrenamiento del modelo de Deep Learning se consigue un 94,75 % de acierto en el test de validación.

Además, se puede observar en la figura 30 el porcentaje de precisión tanto para el dataset de entrenamiento como el de validación, observándose cómo van evolucionando de forma casi idéntica.

En la figura 31 se puede ver la matriz de confusión, aunque esta matriz no es representativa del porcentaje de acierto del modelo dado que las clases no están balanceadas, esto provoca que la diagonal no tenga un número similar de aciertos entre clases, pero es debido a la gran diferencia de muestras de unas clases sobre otras.

En la figura 32 puede observarse el porcentaje de acierto, exhaustividad y f1-score para cada una de las diferentes clases.

## 7.2. iOS App Interpreter American Sign Language

Para la demostración de resultados, se han utilizado algunos videos de la plataforma `http://wwww.youtube.com`. Todas las pruebas se han realizado seleccionando como entrada la cámara del dispositivo, por lo que para el análisis de videos de la plataforma, se proyecta el vídeo en un monitor, de forma que la cámara del dispositivo está enfocada en el monitor y permite realizar el análisis.

También se ha realizado un vídeo para la demostración de resultados del método de funcionamiento Detects words 6.5.

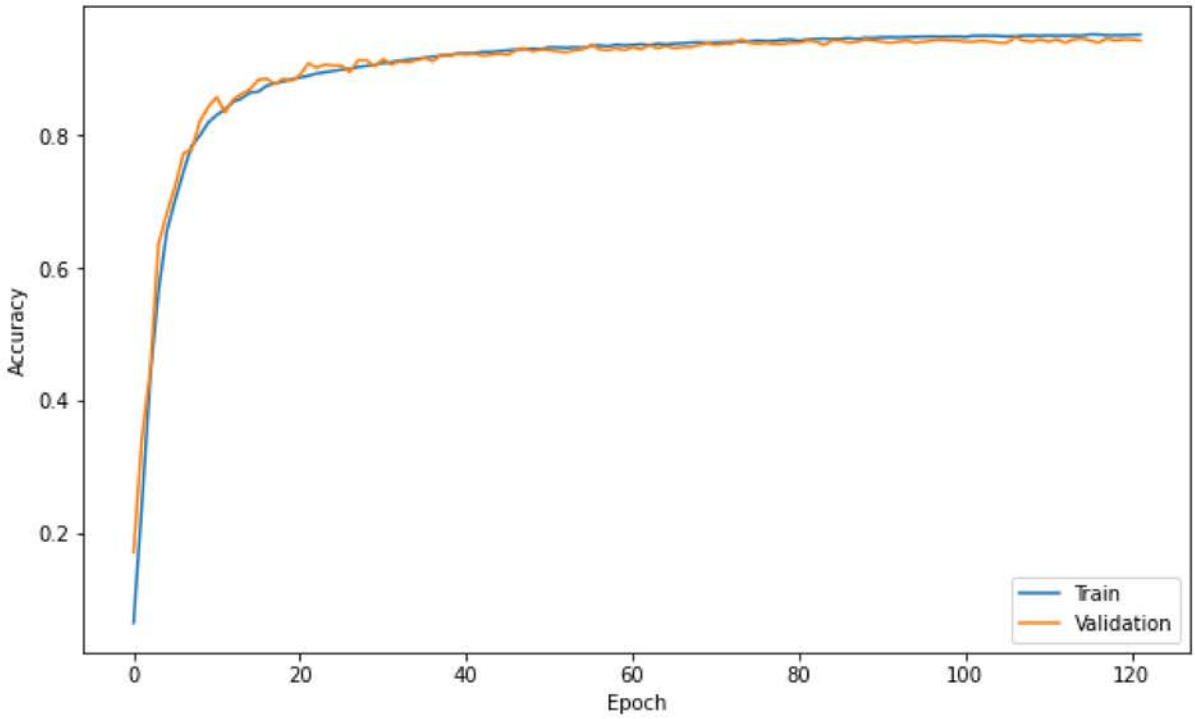



Figura 30: Evolución de la precisión del dataset de entrenamiento y validación.

1. Demo 1 [14]: Se utiliza el método de funcionamiento Letter Occurrences. Se consigue acertar en 23 de 24 letras, únicamente fallando la predicción de la letra 'S'. El video original puede encontrarse en [13].

2. Demo 2 [16]: Se utiliza el método de funcionamiento Letter Occurrences. Se consigue acertar en 22 de 24 letras, fallando en las letras 'N' y 'P'. El vídeo original puede encontrarse en [15]

3. Demo 3 [17]: Se utiliza el método de funcionamiento Detect words y se consigue con éxito acertar todas las letras y distinguir cuando se empieza y acaba una palabra. Este video ha sido realizado para la demostración del TFG.

Para obtener una resultados precisos es necesario que el dispositivo esté situado con un trípode. El entorno en el que se han realizado las pruebas puede verse en [12].

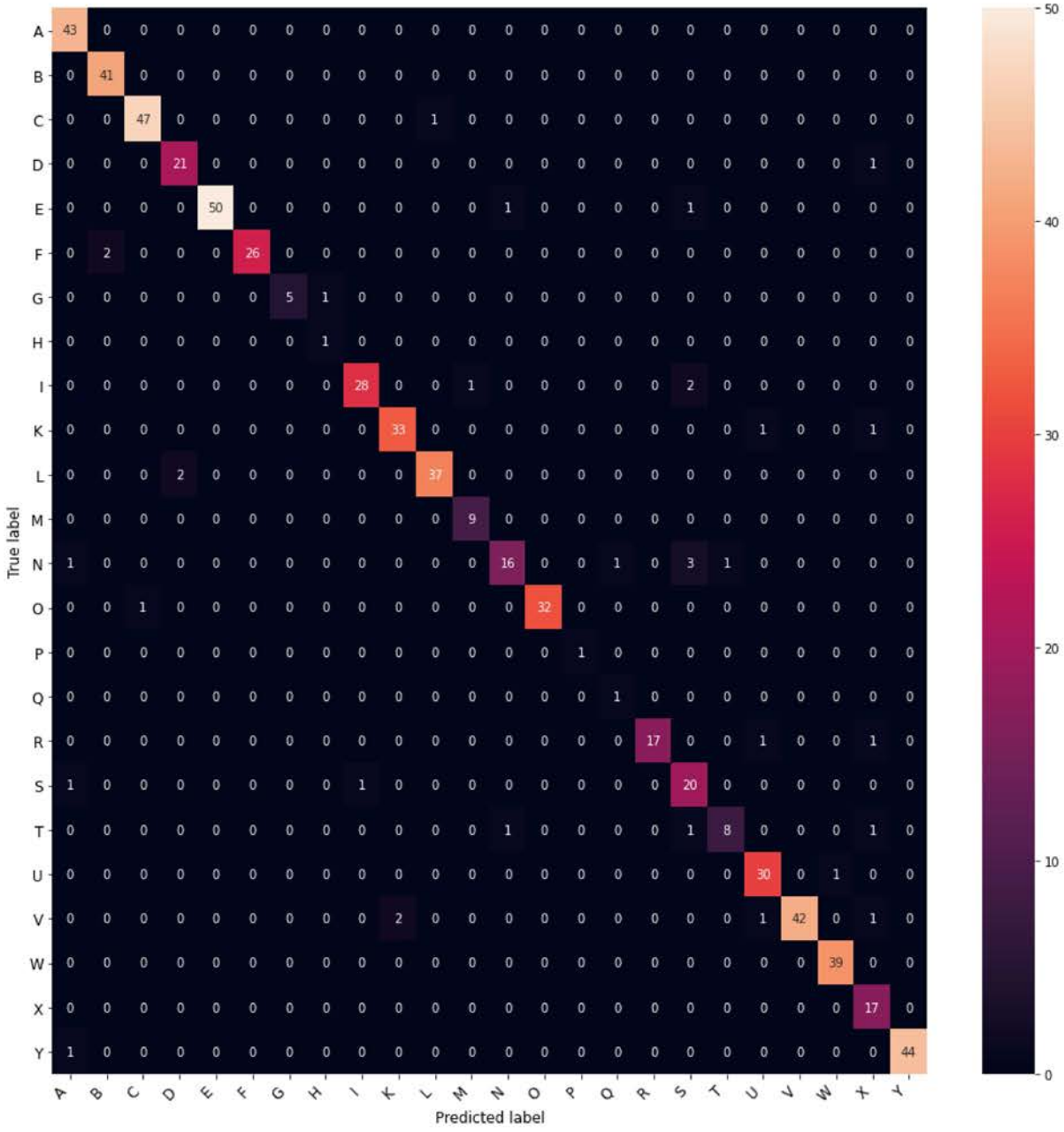



Figura 31: Matriz de confusión.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| A | 0.93 | 1.00 | 0.97 | 43 |
| B | 0.95 | 1.00 | 0.98 | 41 |
| C | 0.98 | 0.98 | 0.98 | 48 |
| D | 0.91 | 0.95 | 0.93 | 22 |
| E | 1.00 | 0.96 | 0.98 | 52 |
| F | 1.00 | 0.93 | 0.96 | 28 |
| G | 1.00 | 0.83 | 0.91 | 6 |
| H | 0.50 | 1.00 | 0.67 | 1 |
| I | 0.97 | 0.90 | 0.93 | 31 |
| K | 0.94 | 0.94 | 0.94 | 35 |
| L | 0.97 | 0.95 | 0.96 | 39 |
| M | 0.90 | 1.00 | 0.95 | 9 |
| N | 0.89 | 0.73 | 0.80 | 22 |
| O | 1.00 | 0.97 | 0.98 | 33 |
| P | 1.00 | 1.00 | 1.00 | 1 |
| Q | 0.50 | 1.00 | 0.67 | 1 |
| R | 1.00 | 0.89 | 0.94 | 19 |
| S | 0.74 | 0.91 | 0.82 | 22 |
| T | 0.89 | 0.73 | 0.80 | 11 |
| U | 0.91 | 0.97 | 0.94 | 31 |
| V | 1.00 | 0.91 | 0.95 | 46 |
| W | 0.97 | 1.00 | 0.99 | 39 |
| X | 0.77 | 1.00 | 0.87 | 17 |
| Y | 1.00 | 0.98 | 0.99 | 45 |
| accuracy | | | 0.95 | 642 |
| macro avg | 0.91 | 0.94 | 0.91 | 642 |
| weighted avg | 0.95 | 0.95 | 0.95 | 642 |

was predicted as 0 0.9520
was predicted as 3 0.8774

Figura 32: Porcentaje de precisión y recall.

# 8

# Conclusiones y Líneas Futuras

## 8.1. Conclusiones

Tras los resultados obtenidos, se puede confirmar que se han cumplido los objetivos:

- Porcentaje de acierto: Se ha conseguido un porcentaje de acierto muy elevado, únicamente teniendo alguna dificultad entre letras cuya representación es muy similar como las letras 'S' y 'T'.
- Rendimiento: las pruebas han sido realizadas en un iPhone 12 Pro, el cual tiene un rendimiento superior al exigido por el proyecto.
- Privacidad y consumo de datos: La aplicación realiza todo el procesamiento en el dispositivo, llevando al máximo la privacidad que puede tener el usuario y al mínimo el consumo de datos.

Por lo tanto, si se utiliza la aplicación en las condiciones indicadas, los resultados son exitosos.

## 8.2. Líneas Futuras

iOS App para la interpretación de Palabras: Este proyecto se ha establecido como un paso previo a la interpretación de palabras. Esto requiere la interpretación de un gesto dinámico, en vez de una imágen estática. Pero se pueden utilizar las bases sentadas en este proyecto para ello. Es necesario que cada entrada del modelo, sea un conjunto de mediciones de entrada, entre las cuales representan un gesto. Por lo tanto, el modelo de Deep Learning realizará predicciones sobre una secuencia de mediciones, que en su totalidad representan una palabra.

# Bibliografía


[1] *A request that detects a human hand pose.* https://developer.apple.com/documentation/vision/vndetecthumanhandposerequest. Accessed: 2021-04-18.

[2] *American Sign Language Alphabet (Static).* https://www.kaggle.com/jordiviader/american-sign-language-alphabet-static. Accessed: 2021-04-18.

[3] *Anaconda.* https://www.anaconda.com/. Accessed: 2021-04-18.

[4] *Apple Vision Framework.* http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm. Accessed: 2021-04-18.

[5] *ASL Alphabet.* https://www.kaggle.com/grassknoted/asl-alphabet. Accessed: 2021-04-18.

[6] *ASL Alphabet Test.* https://www.kaggle.com/danrasband/asl-alphabet-test. Accessed: 2021-04-18.

[7] *ASL Sign Language Alphabet Pictures [Minus J, Z].* https://www.kaggle.com/signnteam/asl-sign-language-pictures-minus-j-z. Accessed: 2021-04-18.

[8] *Clean Swift Arquitecture.* https://clean-swift.com/. Accessed: 2021-04-18.

[9] *Core ML.* https://developer.apple.com/documentation/coreml. Accessed: 2021-04-18.

[10] *Core ML Tools.* https://github.com/apple/coremltools. Accessed: 2021-04-18.

[11] *Deep learning for humans.* https://keras.io/. Accessed: 2021-04-18.

[12] *Demo 1 entorno.* https://drive.google.com/file/d/1nNVTbPlbGVKG3owX-LqEt87ZXOHElRya/view?usp=sharing. Accessed: 2021-04-18.

[13] *Demo 1 Original.* https://www.youtube.com/watch?v=u3HoC9_ir3s. Accessed: 2021-04-18.

[14] *Demo 1.* https://drive.google.com/file/d/1KkUIqCf3Tt-9WOQSwpXr-TYVpAd1zbjk/view?usp=sharing. Accessed: 2021-04-18.

[15] *Demo 2 Original.* https://www.youtube.com/watch?v=jEB45Z6xlAg&t=8s. Accessed: 2021-04-18.

[16] *Demo 2.* https://drive.google.com/file/d/1PGQKLLEMsoQ2uyCajkaIyhuaiSK2gvv_/view?usp=sharing. Accessed: 2021-04-18.

[17] *Demo detect words.* https://drive.google.com/file/d/1WvnkZU0em1m8L8WMGAa5OwkNgFxIsxPM/view?usp=sharing. Accessed: 2021-04-18.

[18] *Detect Body and Hand Pose with Vision WWDC 2020.* https://developer.apple.com/videos/play/wwdc2020/10653/. Accessed: 2021-04-18.

[19] *Django REST framework is a powerful and flexible toolkit for building Web APIs.* https://www.django-rest-framework.org/. Accessed: 2021-04-18.

[20] *IDE de Python para desarrolladores profesionales.* https://www.jetbrains.com/es-es/pycharm/. Accessed: 2021-04-18.

[21] *Jupyter.* https://jupyter.org/. Accessed: 2021-04-18.

[22] *Significant (ASL) Sign Language Alphabet Dataset.* https://www.kaggle.com/kuzivakwashe/significant-asl-sign-language-alphabet-dataset. Accessed: 2021-04-18.

[23] *TensorFlow es una plataforma de código abierto de extremo a extremo para el aprendizaje automático.* https://www.tensorflow.org/. Accessed: 2021-04-18.

# Apéndice A
# Jupyter Notebook

# aslExtractedFeaturesTFGNotebook

April 24, 2021

[1]:
```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import *
from keras import optimizers, callbacks
import keras.backend as k

%matplotlib inline
import matplotlib.pyplot as plt

import sys, math
import pandas as pd
from sklearn import preprocessing
```

```
Using TensorFlow backend.
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:516: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:517: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:518: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing
```

```
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: FutureWarning:
Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

[2]:
```
labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "K",
          "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U",
          "V", "W", "X", "Y"]
num_classes = len(labels)

data_dir = "Dataset_short_v3"
train_dir = os.path.join(data_dir, "train")
batch_size = 32
```

```
# X: features list, Y: class list to predict
X = []
Y = []

# read each folder of train
for label in labels:
    name = os.path.join(train_dir, label, label + ".csv")
    df = pd.read_csv(name)
    X.extend(df.values)
    newSize = len(df)
    print("Letter %s: num_data: %d" %( label, newSize))
    for i in range(0,newSize):
        Y.append(label)

# convert to np array to have shape
X = np.array(X)
Y = np.array(Y)

#one hot encoding
lb = preprocessing.LabelBinarizer()
lb.fit(labels)
Y = lb.transform(Y)
```

```
Letter A: num_data: 793
Letter B: num_data: 821
Letter C: num_data: 966
Letter D: num_data: 585
Letter E: num_data: 918
Letter F: num_data: 628
Letter G: num_data: 70
Letter H: num_data: 31
Letter I: num_data: 698
Letter K: num_data: 641
Letter L: num_data: 810
Letter M: num_data: 283
Letter N: num_data: 285
Letter O: num_data: 626
Letter P: num_data: 22
Letter Q: num_data: 11
Letter R: num_data: 544
Letter S: num_data: 546
Letter T: num_data: 260
Letter U: num_data: 595
Letter V: num_data: 768
Letter W: num_data: 818
Letter X: num_data: 347
```

```
Letter Y: num_data: 770
```

[3]:
```
sizeX = len(X)
sizeY = len(Y)
print(X.shape, Y.shape)
```

```
(12836, 42) (12836, 24)
```

[4]:
```
#split into training,val data. test_data
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.05)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.
 ↪05)
```

[5]:
```
print("train data ", X_train.shape, Y_train.shape)
print("val data ", X_val.shape, Y_val.shape)
print("test data ", X_test.shape, Y_test.shape)

#handle unbalance data
Y_train_classes = lb.inverse_transform(Y_train)
len(Y_train_classes)
from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(Y_train_classes),
                                                  Y_train_classes)
class_weight_dic = dict(enumerate(class_weights))
```

```
train data  (11584, 42) (11584, 24)
val data  (610, 42) (610, 24)
test data  (642, 42) (642, 24)
```

```
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/sklearn/utils/validation.py:72: FutureWarning: Pass classes=['A' 'B'
'C' 'D' 'E' 'F' 'G' 'H' 'I' 'K' 'L' 'M' 'N' 'O' 'P' 'Q' 'R' 'S'
 'T' 'U' 'V' 'W' 'X' 'Y'], y=['D' 'C' 'T' … 'O' 'A' 'R'] as keyword args. From
version 1.0 (renaming of 0.25) passing these as positional arguments will result
in an error
  "will result in an error", FutureWarning)
```

[6]:
```
model = Sequential()
model.add(Dense(32, activation='relu', input_shape=(42,)))
model.add(Dense(num_classes))
model.add(Activation("softmax"))
```

```
WARNING:tensorflow:From
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/keras/backend/tensorflow_backend.py:74: The name tf.get_default_graph
is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

```
WARNING:tensorflow:From
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/keras/backend/tensorflow_backend.py:517: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/keras/backend/tensorflow_backend.py:4138: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.
```

[7]:
```
model.summary()
model.compile(optimizer=optimizers.SGD(lr=0.01),␣
 ↪loss="categorical_crossentropy",
                  metrics=["accuracy"])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                1376
_________________________________________________________________
dense_2 (Dense)              (None, 24)                792
_________________________________________________________________
activation_1 (Activation)    (None, 24)                0
=================================================================
Total params: 2,168
Trainable params: 2,168
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/keras/optimizers.py:790: The name tf.train.Optimizer is deprecated.
Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From
/home/dani/anaconda3/envs/kerasenvTFGExtractedFeatures/lib/python3.6/site-
packages/keras/backend/tensorflow_backend.py:3295: The name tf.log is
deprecated. Please use tf.math.log instead.
```

[8]:
```
checkpoint_dir = "checkpointsASLFeaturesTFGNotebook/"
checkpoint_name = (checkpoint_dir
    + "featuresTFGNotebook-{val_loss:.4f}-{val_acc:.4f}.hdf5")

if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)
```

```
def create_callbacks():
    return [
        callbacks.EarlyStopping(
            monitor="val_acc",
            patience=15,
            verbose=1),
        callbacks.ModelCheckpoint(
            checkpoint_name,
            monitor="val_acc",
            verbose=1,
            save_best_only=True),
    ]

my_callbacks = create_callbacks()
histories = []
```

[10]:
```
#finetuning
K.set_value(model.optimizer.lr,
                K.get_value(model.optimizer.lr) / 2)

histories.append(model.fit(
    X_train,
    Y_train,
    batch_size=batch_size,
    epochs=200,
    validation_data=(X_val, Y_val),
    callbacks=my_callbacks,
    class_weight=class_weight_dic))
```

```
Train on 11584 samples, validate on 610 samples
Epoch 1/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2473 -
acc: 0.9465 - val_loss: 0.2515 - val_acc: 0.9426

Epoch 00001: val_acc did not improve from 0.94426
Epoch 2/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2451 -
acc: 0.9475 - val_loss: 0.2522 - val_acc: 0.9410

Epoch 00002: val_acc did not improve from 0.94426
Epoch 3/200
11584/11584 [==============================] - 0s 28us/step - loss: 0.2445 -
acc: 0.9476 - val_loss: 0.2472 - val_acc: 0.9393

Epoch 00003: val_acc did not improve from 0.94426
Epoch 4/200
11584/11584 [==============================] - 0s 30us/step - loss: 0.2433 -
```

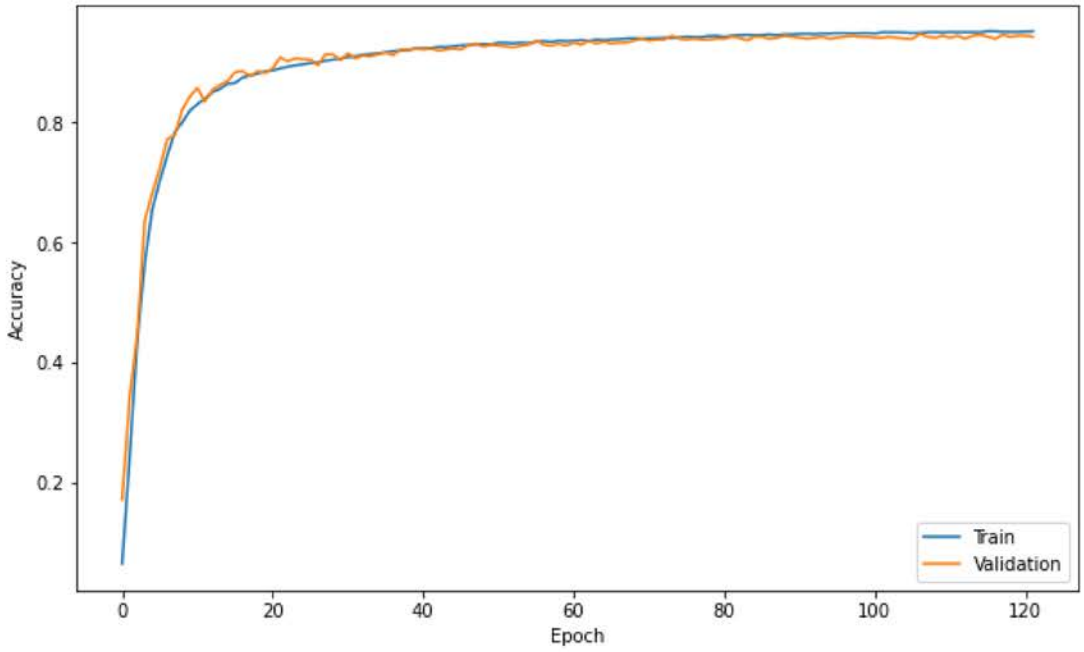
```
acc: 0.9472 - val_loss: 0.2505 - val_acc: 0.9410

Epoch 00004: val_acc did not improve from 0.94426
Epoch 5/200
11584/11584 [==============================] - 0s 28us/step - loss: 0.2427 -
acc: 0.9479 - val_loss: 0.2473 - val_acc: 0.9426

Epoch 00005: val_acc did not improve from 0.94426
Epoch 6/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2425 -
acc: 0.9480 - val_loss: 0.2471 - val_acc: 0.9393

Epoch 00006: val_acc did not improve from 0.94426
Epoch 7/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2414 -
acc: 0.9485 - val_loss: 0.2501 - val_acc: 0.9410

Epoch 00007: val_acc did not improve from 0.94426
Epoch 8/200
11584/11584 [==============================] - 0s 27us/step - loss: 0.2402 -
acc: 0.9485 - val_loss: 0.2469 - val_acc: 0.9426

Epoch 00008: val_acc did not improve from 0.94426
Epoch 9/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2404 -
acc: 0.9483 - val_loss: 0.2450 - val_acc: 0.9443

Epoch 00009: val_acc did not improve from 0.94426
Epoch 10/200
11584/11584 [==============================] - 0s 27us/step - loss: 0.2392 -
acc: 0.9485 - val_loss: 0.2455 - val_acc: 0.9426

Epoch 00010: val_acc did not improve from 0.94426
Epoch 11/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2384 -
acc: 0.9487 - val_loss: 0.2451 - val_acc: 0.9426

Epoch 00011: val_acc did not improve from 0.94426
Epoch 12/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2371 -
acc: 0.9479 - val_loss: 0.2476 - val_acc: 0.9410

Epoch 00012: val_acc did not improve from 0.94426
Epoch 13/200
11584/11584 [==============================] - 0s 27us/step - loss: 0.2373 -
acc: 0.9499 - val_loss: 0.2424 - val_acc: 0.9410

Epoch 00013: val_acc did not improve from 0.94426
```

```
Epoch 14/200
11584/11584 [==============================] - 0s 30us/step - loss: 0.2352 -
acc: 0.9499 - val_loss: 0.2418 - val_acc: 0.9426

Epoch 00014: val_acc did not improve from 0.94426
Epoch 15/200
11584/11584 [==============================] - 0s 29us/step - loss: 0.2358 -
acc: 0.9499 - val_loss: 0.2429 - val_acc: 0.9410

Epoch 00015: val_acc did not improve from 0.94426
Epoch 16/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2346 -
acc: 0.9495 - val_loss: 0.2432 - val_acc: 0.9393

Epoch 00016: val_acc did not improve from 0.94426
Epoch 17/200
11584/11584 [==============================] - 0s 28us/step - loss: 0.2334 -
acc: 0.9488 - val_loss: 0.2473 - val_acc: 0.9393

Epoch 00017: val_acc did not improve from 0.94426
Epoch 18/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2327 -
acc: 0.9492 - val_loss: 0.2425 - val_acc: 0.9475

Epoch 00018: val_acc improved from 0.94426 to 0.94754, saving model to
checkpointsASLFeaturesTFGNotebook/featuresTFGNotebook-0.2425-0.9475.hdf5
Epoch 19/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2331 -
acc: 0.9499 - val_loss: 0.2383 - val_acc: 0.9426

Epoch 00019: val_acc did not improve from 0.94754
Epoch 20/200
11584/11584 [==============================] - 0s 29us/step - loss: 0.2323 -
acc: 0.9503 - val_loss: 0.2400 - val_acc: 0.9410

Epoch 00020: val_acc did not improve from 0.94754
Epoch 21/200
11584/11584 [==============================] - 0s 27us/step - loss: 0.2310 -
acc: 0.9498 - val_loss: 0.2409 - val_acc: 0.9443

Epoch 00021: val_acc did not improve from 0.94754
Epoch 22/200
11584/11584 [==============================] - 0s 28us/step - loss: 0.2306 -
acc: 0.9504 - val_loss: 0.2370 - val_acc: 0.9410

Epoch 00022: val_acc did not improve from 0.94754
Epoch 23/200
11584/11584 [==============================] - 0s 27us/step - loss: 0.2301 -
```

```
acc: 0.9500 - val_loss: 0.2387 - val_acc: 0.9443

Epoch 00023: val_acc did not improve from 0.94754
Epoch 24/200
11584/11584 [==============================] - 0s 27us/step - loss: 0.2290 -
acc: 0.9501 - val_loss: 0.2398 - val_acc: 0.9393

Epoch 00024: val_acc did not improve from 0.94754
Epoch 25/200
11584/11584 [==============================] - 0s 29us/step - loss: 0.2281 -
acc: 0.9501 - val_loss: 0.2378 - val_acc: 0.9443

Epoch 00025: val_acc did not improve from 0.94754
Epoch 26/200
11584/11584 [==============================] - 0s 30us/step - loss: 0.2276 -
acc: 0.9499 - val_loss: 0.2332 - val_acc: 0.9459

Epoch 00026: val_acc did not improve from 0.94754
Epoch 27/200
11584/11584 [==============================] - 0s 27us/step - loss: 0.2276 -
acc: 0.9519 - val_loss: 0.2353 - val_acc: 0.9426

Epoch 00027: val_acc did not improve from 0.94754
Epoch 28/200
11584/11584 [==============================] - 0s 28us/step - loss: 0.2262 -
acc: 0.9517 - val_loss: 0.2375 - val_acc: 0.9393

Epoch 00028: val_acc did not improve from 0.94754
Epoch 29/200
11584/11584 [==============================] - 0s 30us/step - loss: 0.2262 -
acc: 0.9507 - val_loss: 0.2307 - val_acc: 0.9459

Epoch 00029: val_acc did not improve from 0.94754
Epoch 30/200
11584/11584 [==============================] - 0s 30us/step - loss: 0.2251 -
acc: 0.9505 - val_loss: 0.2344 - val_acc: 0.9426

Epoch 00030: val_acc did not improve from 0.94754
Epoch 31/200
11584/11584 [==============================] - 0s 26us/step - loss: 0.2245 -
acc: 0.9507 - val_loss: 0.2315 - val_acc: 0.9443

Epoch 00031: val_acc did not improve from 0.94754
Epoch 32/200
11584/11584 [==============================] - 0s 32us/step - loss: 0.2238 -
acc: 0.9512 - val_loss: 0.2336 - val_acc: 0.9443

Epoch 00032: val_acc did not improve from 0.94754
```

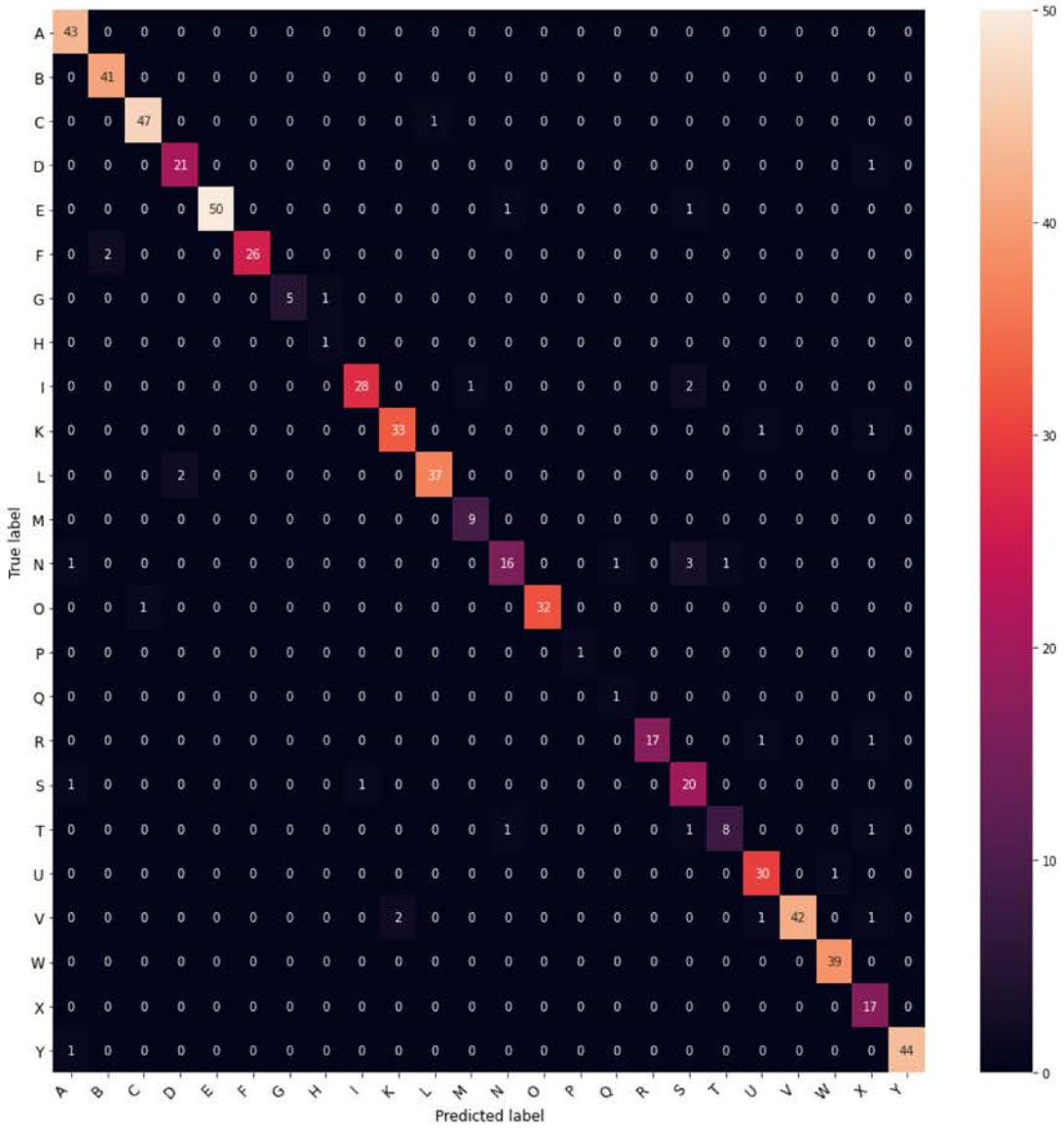
```
Epoch 33/200
11584/11584 [==============================] - 0s 29us/step - loss: 0.2226 -
acc: 0.9517 - val_loss: 0.2321 - val_acc: 0.9426

Epoch 00033: val_acc did not improve from 0.94754
Epoch 00033: early stopping
```

[11]:
```
def combine_histories():
  history = {
      "loss": [],
      "val_loss": [],
      "acc": [],
      "val_acc": []
  }

  for h in histories:
      for k in history.keys():
          history[k] += h.history[k]
  return history

history = combine_histories()

def plot_accuracy(history):
  fig = plt.figure(figsize=(10, 6))
  plt.plot(history["acc"])
  plt.plot(history["val_acc"])
  plt.xlabel("Epoch")
  plt.ylabel("Accuracy")
  plt.legend(["Train", "Validation"])
  plt.show()

plot_accuracy(history)
```




[12]:
```
#get test data in format
probabilities = model.predict(X_test)
predicted_labels = np.argmax(probabilities, axis=-1)

target_labels_classes = lb.inverse_transform(Y_test)

#obtein index of class
le = preprocessing.LabelEncoder()
le.fit(labels)
target_labels = le.transform(target_labels_classes)

from sklearn import metrics
conf = metrics.confusion_matrix(target_labels, predicted_labels)

import seaborn as sns

def plot_confusion_matrix(conf, labels, figsize=(8, 8)):
    fig = plt.figure(figsize=figsize)
    heatmap = sns.heatmap(conf, annot=True, fmt="d")
    heatmap.xaxis.set_ticklabels(labels, rotation=45,
                                 ha="right", fontsize=12)
    heatmap.yaxis.set_ticklabels(labels, rotation=0,
                                 ha="right", fontsize=12)
    plt.xlabel("Predicted label", fontsize=12)
    plt.ylabel("True label", fontsize=12)
```

```
    plt.show()



plot_confusion_matrix(conf, labels, figsize=(16, 16))
```




[13]:
```
#metricas
print(metrics.classification_report(target_labels, predicted_labels,␣
 ↪target_names=labels))

wrong_letters = np.where(predicted_labels != target_labels)[0]
```

```
probs_max = np.max(probabilities, axis=-1)
idx = np.argsort(probs_max[wrong_letters])
idx = idx[::-1][:5]
worst_predictions = wrong_letters[idx]
worst_predictions

worst_predictions

for i in worst_predictions:
    print("was predicted as %s  %.4f" % (
            predicted_labels[i],
            probs_max[i]
    ))
```

```
              precision    recall  f1-score   support

           A       0.93      1.00      0.97        43
           B       0.95      1.00      0.98        41
           C       0.98      0.98      0.98        48
           D       0.91      0.95      0.93        22
           E       1.00      0.96      0.98        52
           F       1.00      0.93      0.96        28
           G       1.00      0.83      0.91         6
           H       0.50      1.00      0.67         1
           I       0.97      0.90      0.93        31
           K       0.94      0.94      0.94        35
           L       0.97      0.95      0.96        39
           M       0.90      1.00      0.95         9
           N       0.89      0.73      0.80        22
           O       1.00      0.97      0.98        33
           P       1.00      1.00      1.00         1
           Q       0.50      1.00      0.67         1
           R       1.00      0.89      0.94        19
           S       0.74      0.91      0.82        22
           T       0.89      0.73      0.80        11
           U       0.91      0.97      0.94        31
           V       1.00      0.91      0.95        46
           W       0.97      1.00      0.99        39
           X       0.77      1.00      0.87        17
           Y       1.00      0.98      0.99        45

    accuracy                           0.95       642
   macro avg       0.91      0.94      0.91       642
weighted avg       0.95      0.95      0.95       642

was predicted as 0  0.9520
was predicted as 3  0.8774
```

```
was predicted as 8  0.8734
was predicted as 7  0.8679
was predicted as 11  0.8001
```

[14]:
```
import coremltools
from keras.models import load_model
```

[15]:
```
best_model = load_model(checkpoint_dir + "featuresTFGNotebook-0.2425-0.9475.
↪hdf5")
```

[16]:
```
coreml_model = coremltools.converters.keras.convert(
        best_model,
        input_names="handpoint",
        output_names="labelProbability",
        predicted_feature_name="label",
        class_labels=labels)

# add metadata to the model
coreml_model.author = "Daniel Gallego Peralta"
coreml_model.license = "Public"
coreml_model.short_description = "Hand points classifier for 24 different␣
↪letters of ASL"

coreml_model.input_description["handpoint"] = "normalized coordinates for hand␣
↪points"
coreml_model.output_description["labelProbability"]= "Prediction probabilities"
coreml_model.output_description["label"]= "Class label of top prediction"

coreml_model.save("ASLHandPointTFG.mlmodel")
```

```
0 : dense_1_input, <keras.engine.input_layer.InputLayer object at
0x7f112df6b550>
1 : dense_1, <keras.layers.core.Dense object at 0x7f112df6b518>
2 : dense_1__activation__, <keras.layers.core.Activation object at
0x7f117c188cf8>
3 : dense_2, <keras.layers.core.Dense object at 0x7f112df6b940>
4 : activation_1, <keras.layers.core.Activation object at 0x7f112df72668>
```

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA